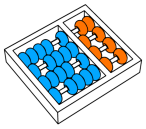
Universidade Estadual de Campinas
Instituto de Computação



Jael Louis Zela Ruiz

Conflict mapping between QoS attributes in SOA Application Monitoring

Mapeamento de Conflitos entre Atributos de QoS na Monitoração de Aplicações SOA

CAMPINAS
2016

Jael Louis Zela Ruiz

Conflict mapping between QoS attributes in SOA Application Monitoring

Mapeamento de Conflitos entre Atributos de QoS na Monitoração de Aplicações SOA

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientadora: Profa. Dra. Cecília Mary Fischer Rubira

Este exemplar corresponde à versão final da Dissertação defendida por Jael Louis Zela Ruiz e orientada pela Profa. Dra. Cecília Mary Fischer Rubira.

CAMPINAS
2016

Agência(s) de fomento e nº(s) de processo(s): CNPq, 162280/2015-7

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

Z375c

Zela Ruiz, Jael Louis, 1988-
Conflict mapping between QoS attributes in SOA application monitoring / Jael Louis Zela Ruiz. – Campinas, SP : [s.n.], 2016.

Orientador: Cecília Mary Fischer Rubira.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Monitoramento online. 2. Serviços Web. 3. Arquitetura orientada a serviços (Computação). 4. Qualidade de serviço (Redes de computadores). I. Rubira, Cecília Mary Fischer,1964-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Mapeamento de conflitos entre atributos de QoS na monitoração de aplicações SOA
Palavras-chave em inglês:
Online monitoring
Web services
Service-oriented architecture (Computer science)
Quality of service (Computer networks)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Cecília Mary Fischer Rubira [Orientador]
Eliane Martins
Delano Medeiros Beder
Data de defesa: 07-12-2016
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Jael Louis Zela Ruiz

Conflict mapping between QoS attributes in SOA Application Monitoring

Mapeamento de Conflitos entre Atributos de QoS na Monitoração de Aplicações SOA

Banca Examinadora:

- Profa. Dra. Cecília Mary Fischer Rubira
  Instituto de Computação, UNICAMP (Orientadora)
- Prof. Dr. Delano Medeiros Beder
  Departamento de Computação, UFSCar
- Profa. Dra. Eliane Martins
  Instituto de Computação, UNICAMP

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 07 de dezembro de 2016

Agradecimentos

Primeramente agradezco a Dios y a toda mi familia por su constante apoyo y por la fuerza que me brindan cada semana.

Agradeço à minha orientadora, professora Cecilia Rubira, por ter me orientado, pela dedicação, compreensão, confiança, paciência, amizade e pela oportunidade que ela me deu. Muito Obrigado!

Agradeço à professora Eliane Martins, pela amizade e compreensão, pela oportunidade de ser seu PED e pelos ensinamentos durante esse tempo. Obrigado professora.

Agradeço à banca examinadora, ao professor Delano Beder e à professora Eliane Martins, pela disponibilidade e valiosas sugestões que me ajudam melhorar. Muito Obrigado.

Agradeço aos meus amigos do IC e da UNICAMP em geral, pela sua amizade e apoio; especialmente aos meus amigos do LSD.

Agradeço ao Rômulo Franco pela sua amizade e orientação no começo do meu mestrado.

Agradeço aos professores do IC, pelos seus ensinamentos.

Agradeço ao professor João Meidanis por me dar a oportunidade de trabalhar na Scylla Bioinformatica, que me ajudou a continuar no meu mestrado.

Agradeço à secretaria Cristiane Camargo, pela orientacão e ajuda nos temas administrativos durante meu mestrado.

Agradeço à UNICAMP que me proporcionou a grande oportunidade.

I would like to thank the professors Chouki Tibermacine and Marianne Huchard, for their help, collaboration, orientation and friendship during my stay in Montpellier, France. I have learned a lot. Thank you so much!.

Agradeço aos órgãos de fomento CNPq e CAPES.

Resumo

O número de serviços Web funcionalmente similares tem crescido na Web, tornando a qualidade do serviço o seu maior diferencial. A fim de medir o nível de qualidade dos serviços Web, ferramentas de monitoramento tem surgido como um componente essencial que coleta valores das métricas definidas para atributos de qualidade com o objetivo de controlar o nível de qualidade do serviço.

Desde que os clientes precisam monitorar mais de um atributo de qualidade ao mesmo tempo, os valores da qualidade do serviço tendem a mudar durante seu monitoramento, podendo produzir conflitos entre eles. É dito que dois ou mais atributos de qualidade estão em conflito quando existe uma interferência, incompatibilidade ou contradição entre eles, produzindo uma degradação nos valores da qualidade de pelo menos um dos atributos. Para dar solução a esse problema, estudos sistemáticos tem sido realizados a fim de construir catálogos de conflitos entre requisitos não-funcionais. Porém, esses catálogos são usados durante a fase de análise e projeto do ciclo de desenvolvimento de software.

Nos últimos anos, serviços REST têm sido usados cada vez mais em sistemas distribuídos modernos. Porém, serviços SOAP ainda são preferidos em sistemas complexos, por causa da segurança oferecida. Esta dissertação realiza uma avaliação experimental, a fim de identificar conflitos potenciais entre atributos de qualidade durante o monitoramento de serviços Web SOAP. Para esse efeito, um experimento e um estudo de caso foram executados, sobre um conjunto de atributos de qualidade pré-selecionados com suas respectivas métricas. Os resultados da avaliação experimental validam conflitos identificados em estudos prévios; porém, encontramos conflitos que não foram mencionados previamente. Foi constado que a causa mais comum de degradação na qualidade dos serviços, foi a falta de memoria para processar quantidades grandes de pedidos.

Abstract

The number of functionally similar web services are increasing on the web, making the quality of service its biggest differential. In order to measure the quality level of web services, monitoring tools have become an essential component, collecting values of metrics to control the quality levels of Web services.

Since customers require to monitor more than one attribute at the same time, quality attributes are prone to change in their values during monitoring time, producing conflicts between them. We define that two or more quality attributes are in conflict when there is an interference, incompatibility or contradiction with each other, producing a degradation in the quality values for at least one attribute. To solve this problem, many systematic studies have been performed in order to construct catalogues about conflicts between non-functional requirements during the analysis and design stages of systems development lifecycle.

Recently, REST services are being used increasingly in modern systems. However, SOAP services are still preferred in large systems because of their provided security. This dissertation conducts a practical experimentation in order to discover potential conflicts between quality attributes during SOAP services monitoring. For this purpose, two experimental evaluations were conducted, an experiment and a case study were conducted using a set of selected quality attributes with their metrics. The results of the experimental evaluations validated the existing conflicts defined in previous studies; additionally, we detected potential conflicts which were not identified in previous studies. It was also identified that the most common cause of degradation of the quality level of Web services, was the lack of memory for processing a lot of requests.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Service-Oriented Architecture (SOA) is an architectural model which provides reusability, agility, loose coupling and interoperability by means of a collection of services [42]. In recent years, Web service has become the most popular and used technology to build SOA applications [60]. Web services are self-contained components, which can be discoverable, reusable, composable, publishable, and located through the Web [4]. They make public their functionality through WSDL (Web Services Description Language) in the case of SOAP services, and URI (Uniform Resource Identifier) in the case of REST services. SOAP services are based on a set of standardized XML technologies, such as WSDL and UDDI (Universal Description, Discovery, and Integration), and on the Internet protocols such as SOAP (Simple Object Access Protocol) and HTTP (Hypertext Transfer Protocol), whereas REST services are based on Web resources manipulated by means of HTTP methods (GET, POST, PUT, DELETE) [14]. The interaction between services is performed by means of exchange of information or data sharing. Additionally, this interaction can also involve the integration of two or more services to accomplish a functionality, which is known as composite services [26]. Due to the Web services' popularity, an increasing number of functionally similar Web services can be found on the Internet [10], which entails to service consumers to ask the question: *"what is the best service?"* or *"which Web service better fits my needs?"* [9]. Service consumers have a difficult task to choose the appropriate service for their requirements.

Quality of Service (QoS) has become the most appropriate criterion to distinguish non-functional requirements between equivalent Web services [60]. QoS is composed by a set of properties or quality attributes, for instance, *availability* defines the probability that the service is operational and accessible for use [29], *performance* defines the speed of the service to complete a request [45], and *integrity* defines the correctness degree of the service to response a request [37]. In addition, quality metrics are defined as an objective measure for quality attributes, and the resulting values of these metrics are known as quality values which represent the quality level of a Web service, for example, the downtime per year for *availability* [25], the number of completed request over a period of time for *performance* [45], or the probability to return false data for *integrity* [27]. A set of quality attributes and their metrics are selected depending on the context and the domain of the service, for example, *availability* and *integrity* can be quality attributes chosen for a banking service, whereas *availability* and *performance* can be quality attributes chosen for

a gaming service. Generally, a set of quality attributes is named as a quality model, and it is usually defined by means of a mutual agreement between service providers and service consumers, resulting in a contract or Service Level Agreement (SLA) [38]. Nevertheless, afterwards a SLA is arranged for both parties, service customers ask a new question: *"Is the defined QoS really satisfied?"*. In order to respond this question, monitoring tools have emerged as an essential component in SLA to collect information about services quality levels. Monitoring tools collect quality values from the Web service at runtime [11]. Currently, there are many monitoring tools developed in the research and industry areas, such as Dynamo [6], Cremona [30], SALMon [1], WebInject [18], SOAP Monitor [17], Webmetrics Web Services Monitoring [55], and FlexMonitorWS [16].

On the other hand, quality attributes present characteristics which make them different from functional requirements. They are *subjective* because they can be interpreted differently by different people; they are *relative* because they depend on the context of the system, and they are *interacting* because they can interfere, conflict or contradict between them [36]. These characteristics combined with the dynamic and unpredictable nature of Web services [28] make the quality attributes to be also in conflict during monitoring time. Two or more quality attributes are in conflict when there is an interference, incompatibility or contradiction with each other, producing a degradation of the quality level for at least one of the attributes. Although monitoring tools are a useful means to collect quality values, they can become an intrusive agent for Web services by creating a stressful environment. Monitoring tools use two strategies to collect quality values, passive monitoring and active monitoring [9]. *Passive monitoring* is a strategy based on sniffing the interaction between the Web service and the customer in order to minimize the interaction with the Web service and the customer. *Active monitoring* is based on sending requests to the Web service; in this strategy, monitoring tools act as a client in order to collect the quality values of the service responses. However, when active monitoring is not properly used, it can overload the Web service and produce a quality degradation on it.

1.1 Context of the Problem

E-commerce has been largely integrated into our lives. Currently, a huge number of purchases are made every day, according to Excelacom[1], Amazon, the most popular online store, has a revenue of $ 203 593 about of 210 items sold per minute. This situation makes online payment to be a fundamental and critical part of every online store in order to support large-scale purchases.

Today, online stores do not need to worry about developing a payment module in their applications, because third-party payment systems are available to be integrated into the online stores using Web services. However, online stores require that payment services meet specific quality requirements. Since online stores are available 24 hours a day throughout the year, they require that the payment service be available to process payments anytime. For this reason, online stores and payment systems establish quality levels to ensure the *availability* of the services. Additionally, online stores integrate mon-

[1]http://www.excelacom.com/resources/blog/2016-update-what-happens-in-one-internet-minute

itoring tools in their systems in order to collect information about the *availability* of the payment services. Generally, monitoring is based on continuously sending request to the Web services, also known as active monitoring, as defined earlier, in order to check if it is ready to process a new payment.

Since, online stores do not only require *availability* in payment services, other attributes are also required to be monitored such as *security*, because payment systems deal with critical information which needs to be protected; *robustness*, since clients can introduce erroneous payment information and the payment system needs to cope with it; *integrity*, in order to ensure the correctness of the transaction and the data; and *performance*, considering that a large number of payments will be supported. In this situation, quality levels for these attributes are also defined by both parties and incorporated in the Service Level Agreement. Consequently, monitoring tools for each quality attribute are added. However, the addition of multiple monitors may incur in conflicts between quality attributes producing degradation of the quality level in one or more quality attributes, due to the applied strategy used to monitor each one of the attributes at the same time, such as monitoring tools.

Previous studies [46, 53] have identified the following causes of quality degradation: incorrect implementation of the business logic, memory leak, deadlock, data race, inconsistent data, lack of resources (CPU, memory, file system, etc), bandwidth, and external agents.

1.2 Definition of the Problem

Conflicts between quality attributes is an important topic in Software Engineering since conflicts are an inevitable characteristic of the interaction of Web services [35]. The use of multiple monitors, each one dedicated to monitor a quality attribute, in an active mode to collect quality values from Web services, increase the interaction with the service and, therefore, a greater likelihood of conflicts among quality attributes can occur. In this work, we define that two or more quality attributes are in conflict during monitoring, when exists an interference between their monitors, producing a degradation in the quality value of one or more attributes. For example, *robustness* can be measured by collecting quality values using fault injection. This technique used to evaluate the failures and errors produced by the Web service by introducing faults. However, fault injection can destabilize the Web service, producing a delay in delivering response to other requests affecting the *Response Time* or stopping the service affecting its *availability*.

Previous studies have studied conflict between Non-Functional Requirements (NFRs) during the analysis and design process in the software development. Mairiza *et al.* [32–36] have conducted an extensive systematic review of the literature in order to collect evidence for potential conflicts between NFRs. As a result, they have constructed a catalogue of conflicts between NFRs (Table 2.3).

On the other hand, Zheng *et al.* [58–60] have conducted a large-scale evaluation of the quality levels of real-world Web services, by means of monitoring quality attributes with the aim to provide an overview of the service of Web services. However, studies

about conflicts between quality attributes during Web service monitoring have not been discussed in the literature. These and other related work are reviewed in more detail in Section 2.2.

1.3 The Proposed Solution

The proposed solution consists of an empirical study composed by two experimental evaluations of Web service monitoring in order to evaluate in practice, potential conflicts between quality attributes. This solution has a secondary aim to validate the conflict catalogue defined by Mairiza *et al.* [33] (Table 2.3).

Our empirical study was performed following the guidelines proposed by Wohlin *et al.* [56] and the quantitative research paradigm. For this purpose, two strategies were adopted based on data collection and statistical analysis. An *experiment* that is conducted in a laboratory environment in order to control and manipulate variables of the study; and a *case study* that is an observational study over an object of study, which we do not have a complete control. For our experiment of evaluation, we have implemented a composite service for a Travel Agency composed of three third-party services. We have also selected a public real Web service as a second object of study. After to define our two objects of study, we selected a quality model composed of quality attributes and quality metrics measurable on Web service at runtime, and a non-intrusive monitoring tool that implements our quality metrics. In both evaluations, active monitoring was used to collect quality values from two scenarios: 1) monitoring quality attributes in isolation, and 2) monitoring quality attributes in pairs with the aim to find conflicts between two quality attributes during monitoring time. In order to summarize and generalize the collected quality values, we have used bootstrapping, a statistical technique to estimate the sampling distribution of our results. We identified conflicts when the distribution has a displacement in its values by comparing the sampling distribution from the two scenarios. Finally, we have also compared the results from our two evaluations and we consolidated them to produce a conflict mapping. Additionally, our mapping was compared with the Mairiza's catalogue in order to validate the identified conflicts.

1.4 Publications

This work has produced an article. This article was an evaluation of a case study for conflicts between the quality attributes *Response Time* and *Accuracy*. The article was published in the journal *Electronic Notes in Theoretical Computer Science*.

- Jael Zela, and Cecília M.F. Rubira. "Quality of Service Conflict During Web Service Monitoring: A Case Study". In Electronic Notes in Theoretical Computer Science, Volume 321, Pages 113-127, 2016.

1.5 Outline

The rest of this work consists of the following chapters:

- **Chapter 2**, *Background and Related Work*, In this chapter, we present the required background for this study; we define SOA, Web services and Monitoring Tools as well as Quality attributes and Quality metrics. In addition, we also detail the main studies regarding conflicts between quality attributes.

- **Chapter 3**, *Proposed Solution*, This chapter describes in detail every step of our proposed solution.

- **Chapter 4**, *Experiment: Travel Agent Service*, The chapter presents our semi-controlled case study, which consists of a composite service constructed by the orquestation of three service for a Travel Agent, conflict between quality attributes are founded during monitoring the Web service.

- **Chapter 5**, *Case Study: Country Info Service*, This chapter presents a real-world case, a public Web service is monitored according to find potential conflicts between quality attributes.

- **Chapter 6**, *Analysis and Discussion*, In this chapter, we analyze the results from our two evaluations and discuss about the main findings. We also present a conflict mapping, result of the aggregation of the two evaluations. Finally, our conflict mapping is compared with previous studies.

- **Chapter 7**, *Conclusions and Future Work*, The chapter presents the final conclusions and contributions of this work, as well as a view of possible future work.

1.6 Final Remarks

In this chapter, we have presented a brief introduction of the problem. Firstly, we introduced the context of the problem, monitoring multiple quality attributes. Secondly, we defined the problem to be solved, conflicts between quality attribute during monitoring SOA applications. Thirdly, we proposed an empirical study by mean of two experimental evaluations. Finally, we present the contributions of this work. In the next chapter, we will review the related work and the background used in the rest of our study.

Chapter 2

Background and Related Work

This chapter defines the fundamental concepts of Web services, Quality of Service and Monitoring Systems used throughout this dissertation in the Section 2.1. On the other hand, Section 2.2 shows an overview of the principal related work for conflicts between quality attributes. A catalogue of conflicts among Non-Functional Requirements is the most representative study during the requirements specification. However, only quality evaluations were found related conflicts between quality attributes during monitoring time.

2.1 Background

In this section, we define briefly the main concepts, models, and techniques which will be used in this work such as SOA, Web services, Monitoring Systems for Web services and Quality of Service. We also describe the main monitoring tools for Web services and essentially FlexMonitorWS Tool which we choose for this study.

2.1.1 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural model widely used in distributed applications, which takes services as the primary functional unit. Services interact with customers through standardized and well-defined interfaces [44]. SOA applications are known for being dynamic, heterogeneous, distributed and autonomous.

SOA implementation technically consists of a combination of technologies, which provide an efficient communication through a common language, they also make the services publishable and discoverable.

Into a SOA environment, there are three fundamental roles: the service provider, the service consumer, and the service broker, as shown in Figure 2.1. The service provider is who designs a software system to be used by end-users through the network by a published and discoverable interface. The service broker is a trusted part that forces to a service provider to meet with privacy laws and regulations to ensure a true relationship between customers and providers. Service broker maintains an index of available services, with an optional additional information about the services including trustworthiness, Quality of Service (QoS), Service Level Agreement (SLA) and possible compensation routes. Service

consumers are organizations, persons, systems or services which interact with the service, they do not need to be concerned about the service implementation, as long as it offers the required functionality and provides the desired quality of service [41, 42].

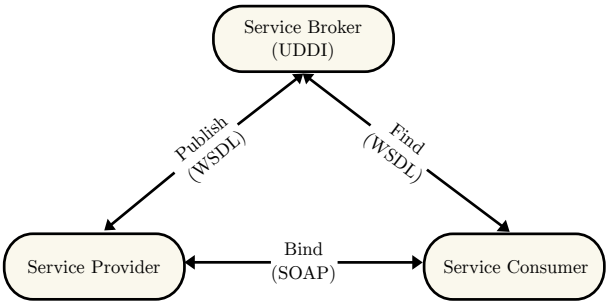


Figure 2.1: SOA Architecture

2.1.2 Web Services

The principal objective of SOA is to represent a reusable unit of business. Papazoglou and Heuvel [42] define a service as a public piece of functionality with three main properties. 1) Services are self-contained, they control their own state. 2) Services are independent of the platform, and this implies that interfaces are not limited to the platform type both on the provider side and customer side. 3) Services in SOA can be dynamically located, invoked and combined.

Web services have become the preferred implementation technology for SOA implementation, promising the maximum service sharing, reuse, and interoperability. Many Web service definitions can be found in the literature, but we consider the definition given by the World Wide Web consortium (W3C) [2]:

> "*A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols*".

Web services are featured for the use of specific Web technologies, such as the Uniform Resource Identifier (URI) to provide a uniform identification, Internet protocols for communication mechanisms, and XML to define the service interface and to represent the exchanged messages. The service provider defines an interface for a service using WSDL (Web Services Description Languages), then it is published in a service broker using a UDDI (Universal Description, Discovery, and Integration) registry, in this way the service is available and discoverable for potential service customers. A service consumer asks the service broker for a particular service, and then the service broker returns the service interface (WSDL) of the most appropriate service. Finally, the service customer invokes the service through SOAP (Simple Object Access Protocol) [41].

2.1.3 Composite Service

A composite service is a set of services and other composite services that collaborate to implement a set of operations [48]. The services that are part of a composite service are referred as component services. An example of composite service would be a travel agent service, integrating service for booking flights, booking hotels, booking cars, etc.

2.1.4 Monitoring Systems

Monitoring has emerged from early of the 1960s, and it was used for different objectives such as debugging, testing, dependability, security, performance and controlling of many kinds of software [52]. Monitoring systems are intended to capture and collect, filter, and analyze information related to the state, behavior, and environment of a software system at runtime [53]. Monitoring is intended to realize a performance analysis, software optimization, software fault detection, diagnosis, and recovery [11]. They are executed in parallel with the software system, without interfering with its normal behavior. A monitor verifies the correctness of a software system by comparing the observed state of the system (collected information) with an expected state (expected information). The information provided for monitoring helps in the decision making for many emerging technologies such as autonomous computing, self-adaptive software, self-managed systems, and fault-tolerance systems [53].

Web Services Monitoring

In a similar form and following the same principles, but with different objectives, Web services also require monitoring. Since monitoring is a fundamental part of SOA system, it is applied in various stages of its life-cycle. For example, monitoring is used during the service discovery process to compute real quality values of Web service and ensure a reliable Web service selection [9]. Then, once a Web service has been selected, monitoring is required to ensure that the Web service fulfills with the promised quality of service.

Since SOA is based on XML-based technologies and communication protocols, the state and behavior of Web services are not only determined by themselves. However, they are also affected by other factors such as hardware, network, client request (i.e., malicious request can produce a deviation of the system behavior) and for other web services (composite services). Thus the deviation in the behavior of a service can produce a deviation of the behavior in the other service.

Web Service Monitoring Model

The Networked European Software and Services Initiative (NESSI) [49] has presented a generic model for monitoring Web service by representing the fundamental principles which are based many monitoring tools. Figure 2.2 shows all common elements of a monitoring system identified by NESSI. The Monitoring Socket represents to the mechanisms to collect Monitoring Datum from a Concrete Service, and this datum is passed through Monitoring Rules to verify some defined Monitoring Constraints. These constraints are based on Quality Metrics to measure the quality of Service Properties, which can refer to

an entire service or Operations. Quality attributes represent a particular type of Service Properties. Additionally, Recovery Actions are trigger when Monitoring Constraints are not fulfilled, and Monitoring Datum can be stored in a History. This whole process is performed for the Service Monitor.

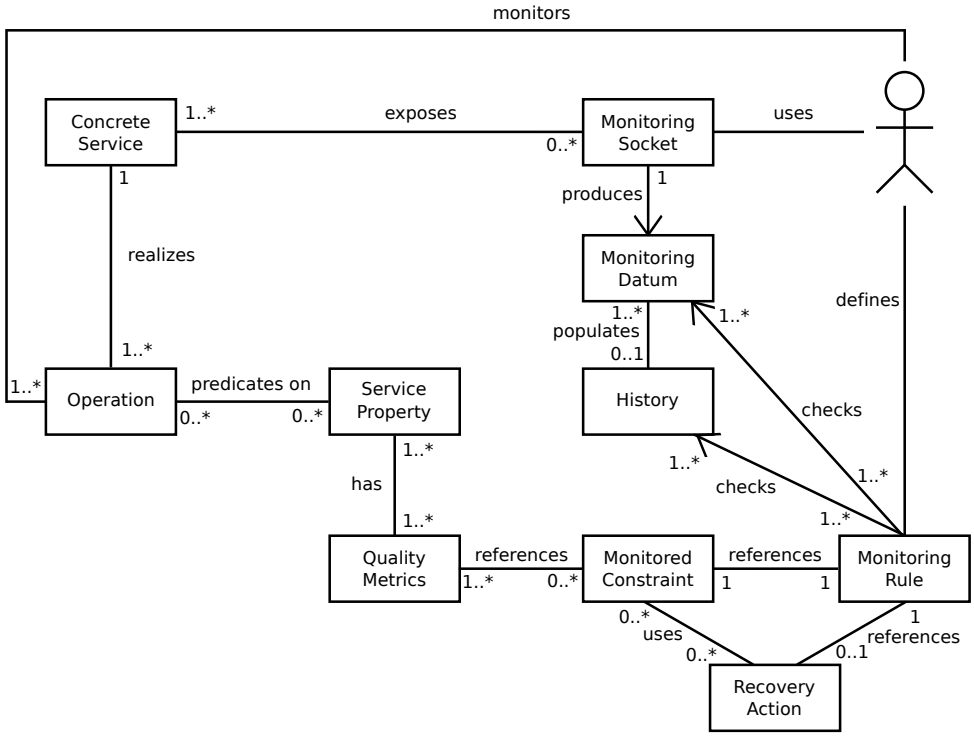


Figure 2.2: Web Service Monitoring Model [49].

Web service Monitoring Taxonomy

Since there are different mechanisms to collect Monitoring Data and various Quality Metrics to measure different Service Properties (QoS), Franco [15] has proposed a taxonomy for Web service monitoring, shown in Figure 2.3. The taxonomy is based on five principal characteristics: Monitoring Target, Quality Attributes, Operation Mode, Monitoring Frequency, and Notification Mode.

1. **Monitoring Target.** Since different factors influence in the proper functioning of Web services, various critical points are identified in a SOA environment which need for monitoring. Figure 2.4 shows the principal monitoring target. 1) The *Web Service* program is often perceived as the most critical point in SOA, and providers tend to spend too much time developing the perfect Web service. 2) The *Container Application* refers to the platform responsible for managing and assign resources for each hosted Web service. It also provides an implementation for common functionalities, for instance, Apache Axis2[1], Glassfish[2] and Apache Tomcat[3]. Container applications are also susceptible to faults that can impact in

[1] `http://axis.apache.org/axis2/java/core/index.html`
[2] `https://glassfish.java.net/`
[3] `http://tomcat.apache.org/`

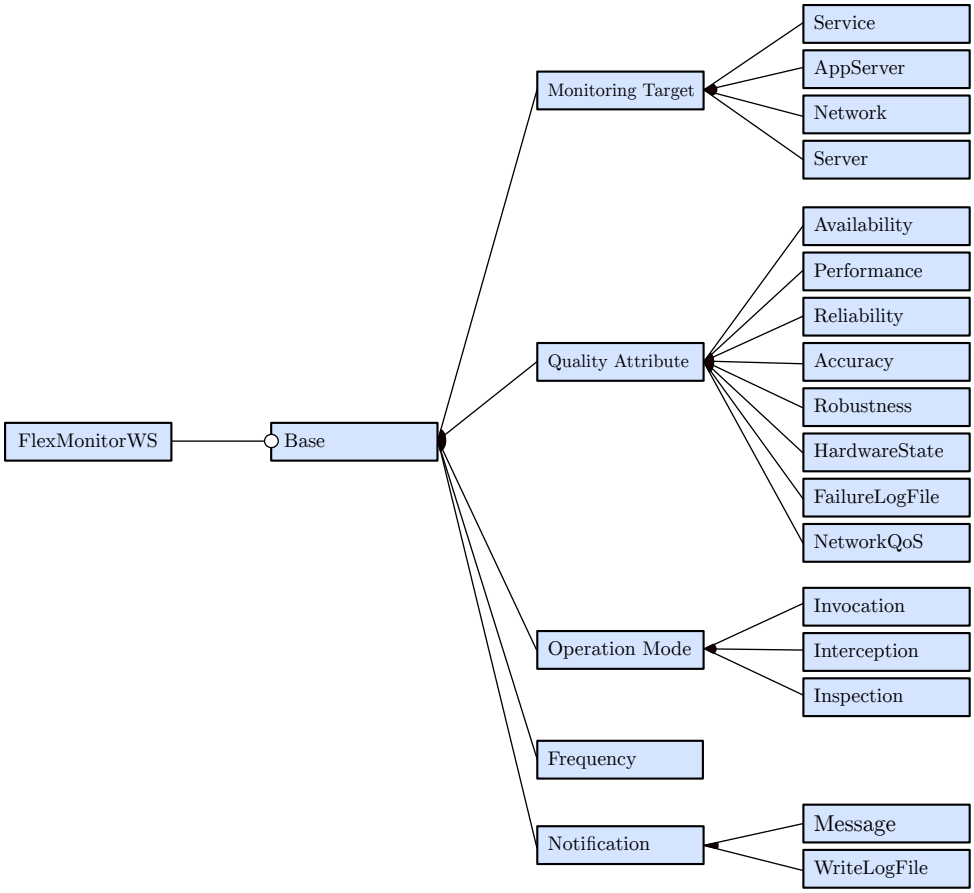


Figure 2.3: FlexMonitorWS Feature Model [15]

all contained services. 3) The *Server*, as a hardware resource, is another critical point, since the hardware is prone to failures and monitoring may include disk drives, memory, so forth. 4) The *Network*, message Exchange between the Web service and the consumer are transmitted through the *Network*, which is exposed to vulnerabilities or bandwidth issues.

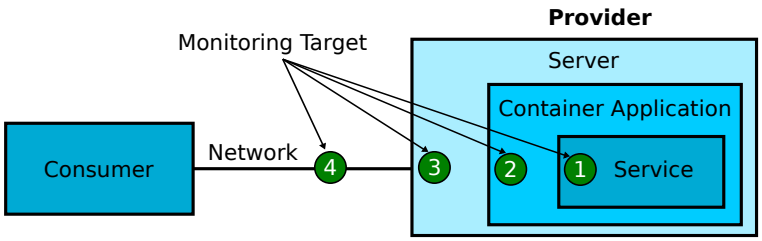


Figure 2.4: Monitoring Targets [15]

On the other hand, Brittenham [8] identified four kinds of configurations for monitoring tool based on three components: Service Consumer, Web Service, and Monitor. Figure 2.5 shows the configurations:

- Configuration 1: The service consumer, the monitor, and the Web service belong to the same system.

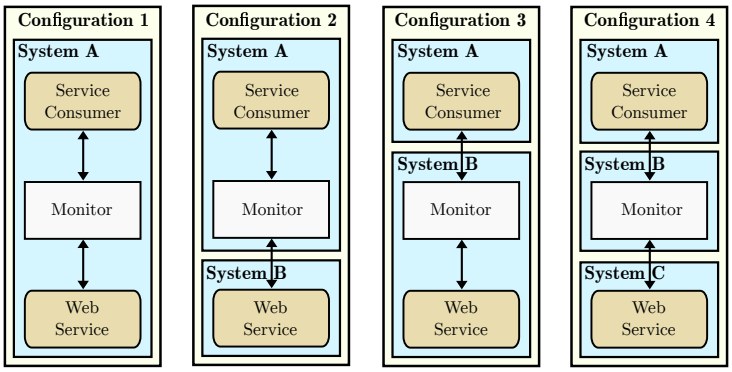


Figure 2.5: Monitor configurations [9].

- Configuration 2: The service consumer and the monitor reside in the same system, and the Web service belongs to a separate system.
- Configuration 3: The service consumer resides in a system, and the monitor and Web service belong to a separate system.
- Configuration 4: The service consumer, the monitor and the Web service reside in different systems.

2. **Quality Attributes.** The primary activity for monitoring tools is to collect monitoring data from a concrete monitoring target, so it is required to define a set of quality attributes for monitoring. Monitoring tools commonly tend to focus on a small set of quality attributes, such as throughput, availability and response time. Quality attributes will be defined in depth in Subsection 2.1.6.

3. **Operation Mode.** In order to collect values of metrics for quality attributes, monitoring tools have used different methods to obtain that values from the Web services. Three methods are most commonly used by monitoring tools [15]. *Interception*, quality values are collected from the messages exchanged between the Web service and the customer. *Invocation*, the monitor acts as a customer and send requests directly to the Web service, with the aim to obtain quality values from the responses. *Inspection*, monitoring is based on to analyze log files to discover an anomalous behavior.

4. **Frequency.** Monitoring can be continuous or periodical. *Continuous*, monitoring is constantly collecting quality values. *Periodical*, monitoring only collects quality values occasionally between predefined intervals of time.

5. **Notification.** The collected quality values can be stored in disk or they can be sent to a third party each interval time or when the monitoring is finished.

2.1.5 Quality of Service

Quality of Service is defined to express non-functional characteristics of a system, in most cases related to the network, but it also applies to servers, databases, applications and distributed systems [45]. The ISO 8402 defines quality as *"the totality of features and*

characteristics of a product or service that bear on its ability to satisfy stated or implied needs". In Web services, we define it as a set of non-functional characteristics that can impact the quality of the Web service.

Oriol *et al.* [39] present an explicit hierarchical relationship between the different concepts in quality of services. Figure 2.6 shows a representation of these concepts.

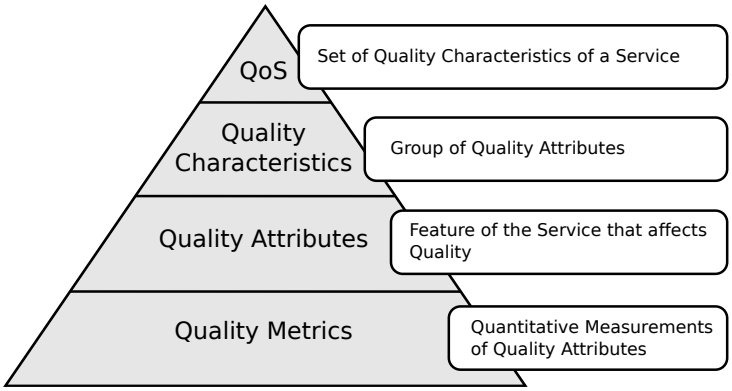


Figure 2.6: Quality Concepts Hierarchy [39].

Quality Characteristics

Quality Characteristics or Quality Factors are properties that not provide quantitative or qualitative measurements. They are high-level quality properties that represent a generic quality attribute for a service [39]. Quality characteristics need to be decomposed into fine-grained concepts, quality attributes. For example, performance, dependability, and security.

Quality Attributes

Quality attributes are features that affect the product or the quality of the service, this includes the variation of its quality values or user experience [50]. Quality attributes are properties that can be quantitatively or qualitatively measurable. They represent an area of concern which may produce a potential impact on the product or service. For example, accuracy, response time, throughput, availability, and integrity.

Quality Metrics

The ISO 24765 [23] defines quality metric as *"a quantitative measurement of the degree to which an item possesses a given quality attribute".* A quality attribute may have several quality metrics depending on the user interests. For example, for Availability, we can define the metrics, Average Availability, Downtime Per Year, and Mean Time Between Failure.

2.1.6 Quality Models

According to Oriol [39], a quality model is a structured set of quality characteristics, which the aim is to group quality attributes for specific purposes, for instance, Web services selection, Web services discovery, Web services monitoring, Ranking of Web services, or Web service composition. However, many other quality models for Web services are proposed by standards, profit and non-profit organization, projects and research authors, by taking a different point of views, approaches or objectives [4, 9, 21, 39–41].

Since Web services are monitored by customers after they are published, it is not possible to extract quality values for every quality attribute. Ran [45] proposed a quality model which brings together the attributes which are measurable in Web service at runtime.

Accuracy

Accuracy is defined as the level of accurate results that the Web service can respond for a quantity of requests [44]. *Accuracy* is measured by the number of invalid responses (error rate) produced by the Web service in a period [15, 29], it is also concerned about the correctness of the responses.

Availability

A Web service should always be ready to be used. *Availability* is defined as the probability that the Web service is up and reached via the network [29] during a period of time.

Capacity

Capacity is the limit of simultaneous or concurrent requests that the Web service can support with guaranteed performance [45].

Latency

Latency is the time taken between a request arrives, and the request is being serviced [45]. There are two types of latency. *Connection Latency* is the time required to establish a connection. *Request Latency* is the time to complete the data transfer after the connection is established. Customers perceive the sum of the connection and request latency [28].

Response Time

Response Time is defined as the required time in response a request [39], and it is the time between sending a request to the service and receiving the corresponding response. However, *Response Time* has a dependency with the network delay and the server side latency.

Reliability

Reliability is defined as the ability of the service to perform its required functions under stated conditions for a specified time interval [29]. Some metrics to measure *Reliability* are the Mean Time Between Failure (MTBF), Mean Time to Failure (MTF), and Mean Time To Transition (MTTT).

Robustness

Robustness is the degree to which a Web service can function correctly even in the present of invalid, incomplete or conflicting inputs [29]. There are many techniques to evaluate the *Robustness*, according to Ran [45], a mechanism to measure *Robustness* is by means of faults injection.

Scalability

Scalability is the capability of the Web service of increasing its computing capacity to process a growing number of requests in a given period. This attribute is also related to performance [28, 44, 45].

Throughput

Throughput is the total number of completed requests in a given period of time. *Throughput* is related to *Latency* and *Capacity* [45]

2.1.7 FlexMonitorWS Tool

Today, a variety of Web service monitoring tools can be found in the literature, and they apply different approaches and techniques. For example: Dynamo [6] uses monitoring rules annotated in BPEL processes in order to collect quality values and verify if constraints are met. Cremona [30] implements a middleware based on Web service agreements, which collect quality values from both sides (service and customer side) verifying the compliance of the agreements. SALMon [1] is based on monitoring services for SLA violations by means of measure instruments instantiated in the Web service. WebInject [18] is a standalone application which send requests to the Web service in order to collect HTTP responses times and calculate the response time of Web services.

FlexMonitorWS [16] is a standalone application based on monitoring profiles and Software Product Lines approach, which allow the creation of a family of monitors for different points in a Web service and different quality attributes using different modes of monitoring. It was developed in Java language using FeatureIDE, a open-source framework for feature-oriented software development based on Eclipse [51]. FlexMonitorWS tool exploits the property of flexibility using the creation of monitoring profiles which serve to a specific target and user requirements [15].

Monitoring profiles are built according to a feature model (Figure 2.3) which is divided in 1) monitoring target, 2) quality attributes, 3) operation mode, 4) monitoring frequency, and 5) notification mode [15, 16]. Monitoring target specifies where the monitoring takes

place, and they can be *Web service*, *server application*, *server*, and/or *network*. Quality attributes indicate what needs to be monitored conforming to what is sought, like *availability*, *performance*, *reliability*, *accuracy*, *robustness*, *hardware state*, *failures in log file*, and/or *network QoS*. Operation mode establishes the strategy to be used, active or passive. Passive monitoring by means of message *interception*. Active monitoring by making *invocations* to the service directly or *inspection* of log files. Monitoring frequency can be *continuously* or *periodically*. Notification mode setup the method to notify the monitoring generated results, by sending *messages* or writing in a *log file*. According to the selected features is created a product, a monitor (jar file), this is executed using a properties file containing the Web service information.

FlexMonitorWS divide the quality attributes according to specific monitoring target. Table 2.1 shows this classification.

Table 2.1: Results for monitoring *TravelAgent* service in Isolation

Quality attribute	**Server**	**Application Container**	**Service**	**Network**
Accuracy			*	
Availability			*	*
Response Time			*	*
Reliability			*	
Robustness			*	
Hardware State	*			
Failures in LogFile		*		

2.2 Related Work

Non-Functional Requirements (NFR) have received increasing attention as a critical factor in the success or failure of software projects. NFRs are defined from two perspectives [33, 35]: 1) NFR as the description of properties, characteristics, or constraints that a software system must present, for instance, system constraints, business rules, external interfaces, and any other requirement; and 2) NFR as the description of quality attributes that the software system must have, for example, availability, accuracy, scalability, etc. For this dissertation, we define NFR as quality attributes that the software system must accomplish.

Despite NFRs are recognized as critical, they are often careless and poorly understood [32]. On the order hand, software developers do not pay enough attention to NFRs [57]. NFRs are also poorly documented, difficult to be captured, specified and managed because most of the software developers do not have sufficient knowledge about NFRs [32].

Many investigations show that quality attributes are difficult to deal. They are even difficult to model, verify, test, and to be measured [36] because of the natural characteristics of the quality attributes, since they are *subjective*, *relative*, and *interacting* [35, 36]. *Subjective*, because they can be interpreted, and evaluated differently by different people. *Relative*, the interpretation and the importance of the quality attributes vary depending

on the system being developed as well as the interested stakeholders. *Interacting*, because they tend to interfere, conflict, and contradict with other attributes. Particular combinations of quality attributes in systems can produce inevitable trade-offs [19].

Since conflicts between quality attributes are inevitable, there are two main research challenges [35]: 1) understand the nature of the complex relationship among quality attributes and 2) develop techniques to deal with conflicts. Techniques to manage conflicts have been presented in the literature [32], focusing on documentation, categorizing, catalogue [33], or list of potential conflicts [13]; where catalogues represent the interrelationship among quality attributes. However, in the literature, studies related to conflict quality attributes during monitoring have not been found. Currently, studies in the literature deal with conflicts between quality attribute during the requirement specifications, designing, and development process. On the other hand, quality of service evaluations were applied for real-world Web services without any concern about conflicts.

2.2.1 Quality Attributes Conflicts

Quality attributes are recognized as a critical factor in the success of software projects because they address the essential issues of quality. However, quality attributes tend to interfere and conflict with each other, and this conflict is known as one of the key characteristics of quality attributes. In 1996, Boehm and In [20] proposed the WinWin technique, which is a manual approach that resolves conflicts among quality attributes during the specification of requirements. It provides a framework for identifying and negotiate conflicts by means of *win conditions*. WinWin uses the Theory W, which is based on the premise *'make everyone a winner'*, an incremental Spiral Model and a negotiation model. Firstly, stakeholders define their *win conditions*. Secondly, conflicts among win conditions are determined, and an *issue schema* is composed for each conflict. Thirdly, stakeholders prepare possible *option schemas* to address the issue. Finally, stakeholders evaluate the options and converge on a mutual option, and this option is proposed in an *agreement schema* [7]. For example, Figure 2.7 shows the results of WinWin technique applied in six stakeholders (General Public, Interoperator, User, Maintainer, Developer, and Customer) and seven quality attributes. *Dependability*, ability to provide a service that can be trusted [3]. *Interoperability*, ability to which two or more systems can exchange information and use the exchanged information [22]. *Usability*, ability which a system can be used by users to achieve specified goals with satisfaction in a specific context [22]. *Performance*, determine the speed and effectiveness of a system [22]. *Evolvability and Portability*, degree of effectiveness and efficiency which can be transferred from one system to another. *Cost and Schedule*, cost per system invocation and the schedule for its invocations [45]. *Reusability*, degree which a component can be used in more than one system [22].

In 2001, Lundberg *et al.* [31] identified many conflicts between maintainability and performance by observing five large industrial applications over a period of five years. The results have shown that the conflict between maintainability and performance is relative to the system, and between the observed applications three different conflicts scenarios were found: no conflict, not always in conflict, and in conflict. Lundberg *et al.* also

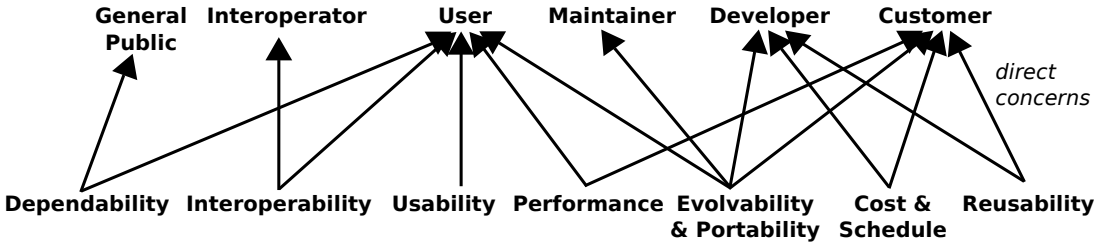


Figure 2.7: Mapping for Stakeholder Interests in Quality Attributes [20].

purposed three ways to handle the conflicts between maintainability and performance: 1) defining methods and guidelines to ensure satisfactory performance without degrading maintainability, 2) implementing techniques that guarantee acceptable performance for object-oriented programs designed for maximum maintainability, and 3) using modern execution platforms which guarantee performance and maintainability.

In 2004, Egyed and Grunbacher [13] conducted a study over the requirements for a simple video-on-demand system, in order to identify conflicts and cooperation between Non-Functional Requirements. For this aim, they proposed an approach which involve 4 activities: 1) manually categorizing requirements into quality attributes, 2) automatically identifying conflicts and cooperation among requirements based on their attributes, 3) automatically generating trace dependencies among the requirements, and 4) filtering out the attributes conflicts and cooperation instances. Table 2.2 shows a conflict mapping between quality attributes obtained by Egyed and Grunbacher, where '+' represents a cooperation between the quality attributes, '−' means a conflict, '+/−' means that a cooperation or conflict between them cannot be determined because both situations were presented in different occasions, and '0' represents that any relationship was found between the quality attributes.

2.2.2 Catalogue of Conflicts among Non-Functional Requirements

In 2009, Mairiza *et al.* [35] presented a review of the literature about managing conflicts among non-functional requirements. In that work, Mairiza identified 106 types of NFRs and outlined the importance of dealing with conflict because they are inevitable. She defined a process for managing NFRs conflicts composed of three main activities: conflict identification, conflict analysis and conflict resolution. Later, in 2010, Mairiza and Zowghi [32] proposed sureCM (security-usability requirement Conflict Management), an ontological framework to deal conflicts between security and usability. The framework receives as input, the security requirements, the system context, the application domain and the usability requirements. Then these inputs are processed through four phases: 1) identify meaning, 2) identify conflicts, 3) characterize conflicts and 4) discover conflict resolution strategies.

In 2011, Mairiza and Zowghi [33] constructed a catalogue of conflicts among 26 types of NFRs based on an extensive and systematic review of the literature. In contrast to Egyed and Grunbacher [13], Mairiza and Zowghi defined three categories of conflicts: absolute conflict, relative conflict and never conflict.

Table 2.2: Model of potential conflict and cooperation requirements [13]

	Functionality	**Efficiency**	**Usability**	**Reliability**	**Security**	**Recoverability**	**Accuracy**	**Maintainability**
Functionality	+	-	+	-	-	0	0	-
Efficiency	0	+/-	+	-	-	0	-	-
Usability	+	+/-	+	+	0	+	+	0
Reliability	0	0	+	+	0	0	0	0
Security	0	-	-	+	+	0	0	0
Recoverability	0	-	+	+	0	+	0	0
Accuracy	0	-	+	0	0	0	+	0
Maintainability	0	0	0	+	+	0	0	+

+ : cooperation relationship
+/- : cooperation or conflict relationship
- : conflict relationship
0 : no relationship

- *Absolute conflict* represents a pair of NFRs that are always in conflict (labeled by 'X').
- *Relative conflict* represents a pair of NFRs that are sometimes in conflict (labeled by '*').
- *Never conflict* represents a pair of NFRs that are never in conflict (labeled by '0').

Table 2.3 presents the conflict catalogue proposed by Mairiza and Zowghi. In this catalogue, we can observe that *Performance* shows conflict with most of the other NFRs, whereas *Dependability* and *Interoperability* only have a conflict with *Performance*.

In 2013, Mairiza *et al.* [34] introduced an experimental approach using the sureCM framework and the conflict catalogue to put in evidence potential conflicts between security and usability. They developed an ontological model which shows conflicts between security and usability, the impacts of the conflicts, and the strategies to resolve the conflict. At the end, they incorporated this approach in the content of the Software Requirements Specification document.

2.2.3 Quality Attributes Conflicts on Monitoring

Depending on how monitoring tools operate over monitored systems, they can produce risk and problems over this last one. Many researchers have reported an intrusiveness problem of monitoring caused by monitoring tools [24, 47, 54]. This problem is due to two main reasons [53]: (1) monitoring tools consume resource, CPU, memory; and (2) potential defects in monitoring tools such as an incorrect implementation or using the inappropriate monitoring technique.

Table 2.3: Catalogue of Conflicts Among NFRs [33]

	Accuracy	**Availability**	**Confidentiality**	**Dependability**	**Flexibility**	**Functionality**	**Interoperability**	**Maintainability**	**Performance**	**Portability**	**Privacy**	**Recoverability**	**Reliability**	**Reusability**	**Robustness**	**Safety**	**Security**	**Testability**	**Understability**	**Usability**
Accuracy	0		*			0		0	X	X		0	0				0			0
Availability									X		X	0				0	X			
Confidentiality	*								X								0			
Dependability									X											
Flexibility																				X
Functionality	0					0		*	*			0	*				*			0
Interoperability									X											
Maintainability	0					*		0	X			0	0		X		0			0
Performance	X	X	X	X		*	X	X	*	X		*	*	X		X	X		X	*
Portability	X								X											
Privacy		X																		
Recoverability	0	0				0		0	*			0	0				0			0
Reliability	0					*		0	*			0	0			0	0			0
Reusability									X											X
Robustness								X								0		X		
Safety		0							X				0		0					
Security	0	X	0			*		0	X			0	0				0			*
Testability															X					
Understability									X											
Usability	0				X	0		0	*			0	0	X			*			0

X : absolute conflict
* : relative conflict
0 : never conflict

Different methods are used to extract and collect information from Web services. These methods are divided into three types: instrument method, interceptor method, and agent approach [53]. *Instrument methods* are used by testing techniques, monitoring code is embedded inside the monitored system, the code is inserted manually by the programmers in different points of the monitored system (e.g. Javassist, AspectJ). *Interceptor methods* are used in the middleware, and they get details about the sent and received messages to the monitored system (e.g. Interceptor in CORBA, Handler in AXIS, JVMTI in JVM). This method is more independent than the *Instrument method*, but it is executed in the same process with the monitored system. *Agent methods* are entirely independent of the monitored system running in its own process [53]. Since *Instrument methods* and *Interceptor methods* are executed in the same location that the monitored system. They use the monitoring configuration 2 (Figure 2.5), whereas *Agent methods* are executed in a different location than the monitored system, it uses the monitoring configuration 4 (Figure 2.5).

Barbacci *et al.* [5] referred to designers to analyze trade-offs between multiples conflicting quality attributes to satisfy user requirements. They recommended that monitoring should not look for a single or universal metric, rather than for individual attributes and trade-off between different metrics. They also recalled that conflict are inevitable. Figure 2.8 shows the opposite directions of the quality attributes, *Performance*, *Dependability*

and *Security*, to reach its optimum level. It is observed that while *Security* reaches its optimum level, the level of *Performance* and *Dependability* decreases. Similar when *Performance* reaches its optimum value, *Dependability* and *Security* reduce their levels. On the other hand, a balanced level is only reached when every quality attribute has a low level.

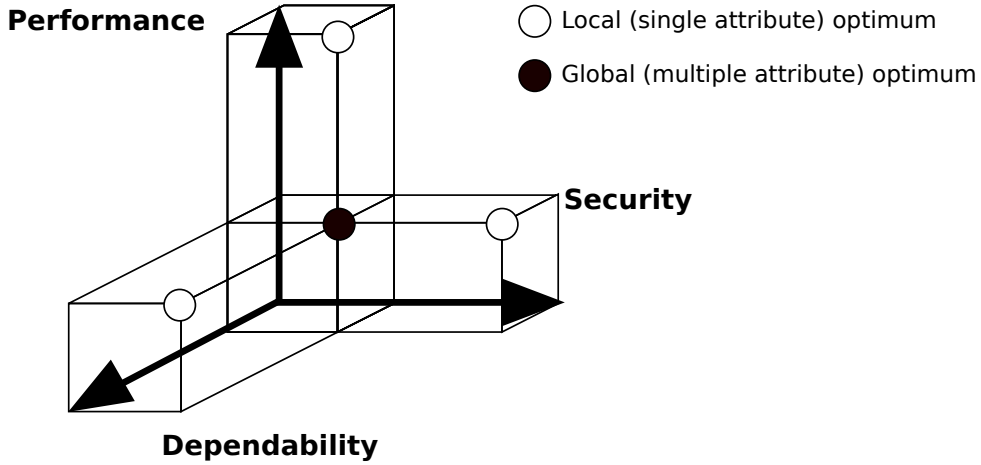


Figure 2.8: Software Quality Attributes Trade-offs [5].

On the other hand, practical experimentations on monitoring Web services can be found in the literature, for example, Zheng *et al.* [58–60] conducted a large-scale distributed evaluation for many real-world Web services. They identified 21,358 Web services located in 89 countries and executed the evaluation using 339 distributed service users. The results show that 77.32% of the WSDL files of the services were available to download, and the 22.68% (4,844) of WSDL files failed for different reasons, the 48.49% were for timeout (Gateway Timeout, Connection timeout, and Read timeout), 30.31% for File Not Found, 10.43% Internal Server Error, and the 10.77% for service unavailable, networks unreachable, unknown host exception, bad request, and bad gateway. The evaluated quality attributes in this experimentation were response time and throughput. The results have shown that a long response time is caused by a long transferring time or a long request processing time. While poor average throughput is caused by poor network conditions in the client-side or server-side.

2.3 Final Remarks

This chapter has presented an overview of the investigation for conflict between quality attributes. Firstly, studies related to conflicts between Non-Functional Requirements were presented during the requirement specification stage in the software development process. Secondly, a conflict catalogue purposed by Mairiza (Subsection 2.2.2) was presented as the most relevant investigation in conflicts between Non-Functional Requirements, but these conflicts are also identified during the software development process. Finally, we presented some practical experiments in monitoring real-world Web services. Nonetheless, no studies related to conflicts between quality attributes during in Web service monitoring were found in the literature. In the following chapter, we present our proposed solution

to obtain a conflict mapping between quality attributes during monitoring Web services. Our proposed solution is based on an experimental evaluation of Web service monitoring in two different scenarios, monitoring quality attributes in isolation, and monitoring in pairs to identify quality level degradation.

On the other hand, the required concepts to understand the rest of this dissertation were defined in Section 2.1. We showed concepts related to Service-Oriented Architecture and Web services as well as concepts of Monitoring Systems and Quality of Service. Finally, we presented FlexMonitorWS, the Web service monitoring tool used in our evaluation study.

Chapter 3

The Proposed Solution

This chapter presents our proposed solution as well as the definition, objectives and planning of the conducted evaluations.

3.1 Overview

Figure 3.1 shows an overview of the activities executed in our experimental process for identifying potential conflicts between quality attributes during monitoring time, as well as the principal artifacts involved in the process. Our experimental evaluations were conducted following the guidelines introduced by Wohlin [56].

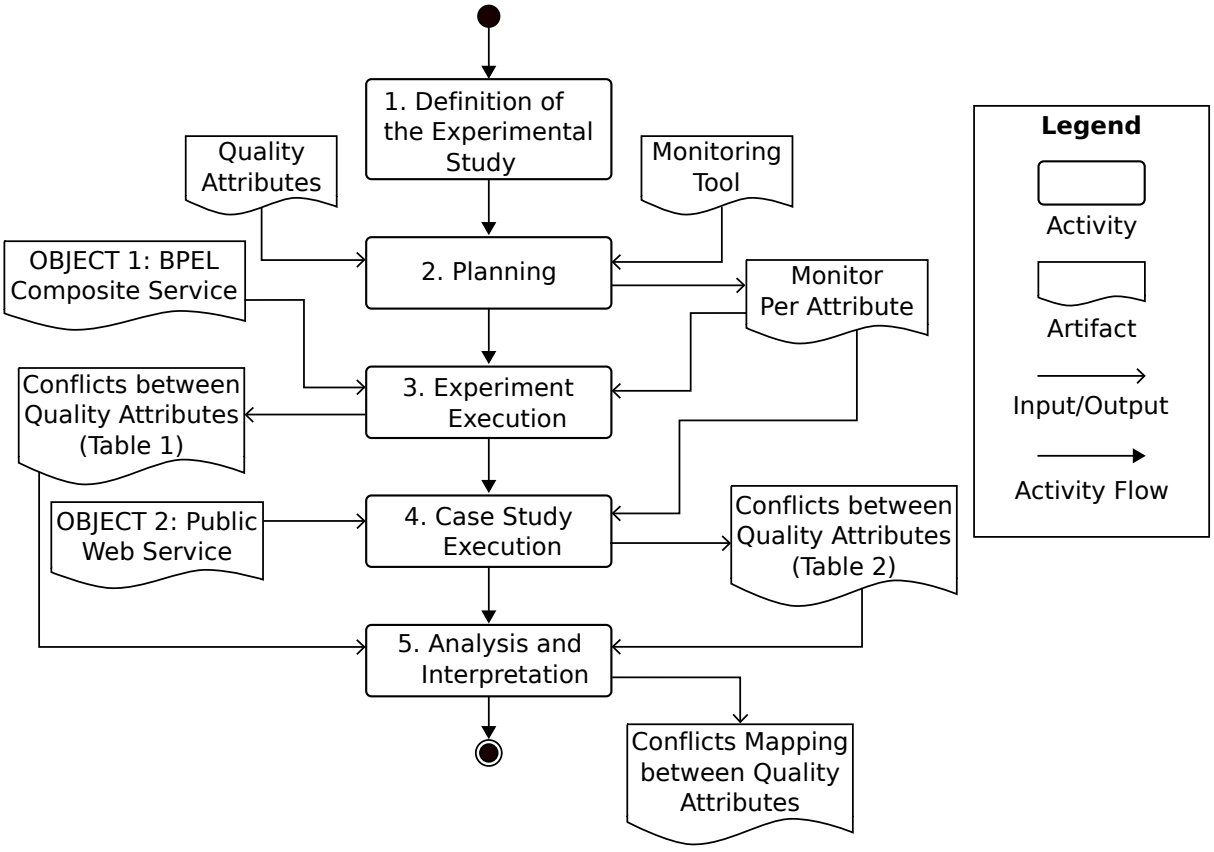


Figure 3.1: Overview of the Experimental Evaluation in UML.

Activities 1 and 2 are exposed in this Chapter, while activities 3, 4, and 5 will be detailed in the Chapters 4, 5, and 6 respectively.

3.2 Activity 1: Definition of the Experimental Study

Our experimental study is motivated by a need to discover and understand conflicts between quality attributes during monitoring. It is known about conflicts between non-functional requirements during their analysis process (Section 2.2); nevertheless, it is also important to understand conflicts between quality attributes during monitoring time. Since an important objective of Web services is to ensure an acceptable quality level to their customers, they tend to use monitoring tools to obtain real values of the quality level of the Web service. However, conflicts between quality attributes during monitoring tend to produce a degradation of the quality of service. For this reason, it is important to understand the interference degree between quality attributes and disclose potential conflicts during monitoring time.

The aim of this study is to analyze the quality attributes for Web services during monitoring time with the purpose of identifying degradation in its quality values concerning the quality attributes. The perspective of this study is taken from the point of view of the service customer in the context of monitoring quality attributes in pairs at the same time.

For this purpose, we conduct two experimental evaluations are conducted following the guidelines for experimental evaluations introduced by Wohlin [56]:

1. *An experiment,* which is done in a laboratory environment, providing a high level of control. The objective of an experiment is to manipulate variables and measure their effects. Statistical analyses are performed on the collected data. This type of evaluation is usually difficult to conduct, since you control many variables of the study. In some cases, it may be impossible to use true experimentation, then the term *quasi-experiment* is used for these experiments.

2. *A case study,* which is used for monitoring projects, activities or assignments. Data is collected for a specific purpose, and statistical analyses are carried out. A case study is an observational study with a low level of control over the experiment. The aim of case studies is to build a model to predict unusual events.

3.2.1 Goals

The goals of the experimental evaluations is:

1. Identify potential conflicts between quality attributes during Web services monitoring.

Our experimental evaluations are conducted following the analysis model GQM (Goal-Question-Metric) [56]. For each goal is formulated one or more questions, and metrics are

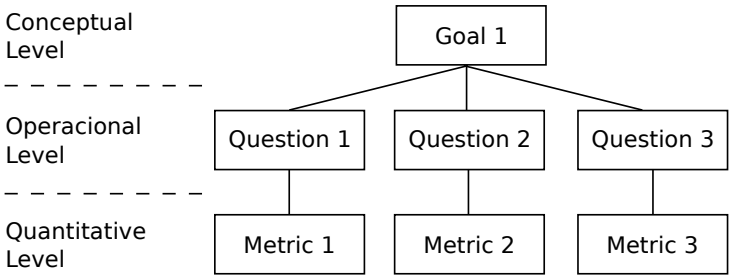


Figure 3.2: GQM Model.

defined for each question in order to validate the results. Figure 3.2 shows the relationship between the Goal, Questions and Metrics for this study.

The questions for our goal are:

1. *What is the quality level of the Web service during monitoring of each quality attribute in isolation?*

2. *What is the quality level of the Web service during monitoring of quality attribute in pairs?*

3. *What are the quality attributes with degraded quality level during monitoring in pairs?*

The metrics for each question are:

1. Confidence interval of the sampling distribution for the quality values collected of monitoring quality attributes in isolation.

2. Confidence interval of the sampling distribution of the quality values collected of monitoring quality attributes in pairs.

3. Difference between the confidence intervals of monitoring in isolation and monitoring in pairs.

The sampling distribution [12] is estimated from the collected quality values, and the quality level for a quality attribute is defined by the confidence interval of the distribution with a confidence factor of 95%. These metrics will be detailed further in Subsection 3.3.1.

3.3 Activity 2: Planning

The context of the two experimental evaluations is monitoring quality attributes. For this context, firstly, it is needed to select a set of quality attributes with their respective metrics. Secondly, it is required to select the adequate monitoring tool.

3.3.1 Hypothesis Formulation

In order to know what we intend to evaluate in the study by means of an experiment and a case study, we formulate our hypotheses as follows:

> Conflict is an inevitable characteristic of interaction, and it has been used to define degradation in the quality values in Web services. We know there are conflicts among non-functional requirements identified during the analysis process of the development phase of a Web service (See catalogue in Subsection 2.2.2), and hence we believe that same conflicts are also presented during Web service monitoring.

Based in this informal statement of the hypothesis, now we state this formally.

> Null Hypotheses, H_0: Two quality attributes which are in conflict during the analysis stage of a Web service development, are also in conflict during the monitoring of Web services.
> H_0: $Conf_{Analysis}(A_i, A_j) = Conf_{Monitoring}(A_i, A_j)$ for $i \neq j$.

3.3.2 Variables Selection

The independent variables for this study are the quality attributes and their metrics, as well as the monitors for each attribute. On the other hand, the dependent variables are the quality values that define the quality level of a Web service.

Quality Attributes

Since Web service are monitored at runtime, we are only interested in quality attributes which are measurable in Web service at runtime. In Subsection 2.1.6, we presented a quality model for monitoring Web service. However, for our experimental evaluations, we selected five quality attributes which can be more easily observable by the service customers. Additionally, we define metrics for each quality attribute.

- **Accuracy** *Accuracy* defines the error rate produced by the Web service, and its metric responds the question, *how many errors does the service produce over a period of time?* [45]

  The metric for *Accuracy* is given by the Equation 3.1, where $nFaults$ is the number of invalid responses returned by the Web service, and $totalRequest$ is the total number of request sent to the Web service.

$$Accuracy = (1 - \frac{nFaults}{totalRequest}) \times 100 \tag{3.1}$$

  The unit of measure for this quality attribute is in percentage. For example, the service *'A'* is 90% accurate. When the quality value for *Accuracy* is closer to 100%, the service is more accurate, but if it is closer to zero percent, the service can lose credibility between the users.

- **Availability** *Availability* is the probability of the Web service to be up and ready to respond a request. Usually, the metric of *Availability* responses the question, *What is the probability that the service responds to my request?* [45]

  The metric for *Availability* is given by the Equation 3.2.

$$Availability = (\frac{upTime}{totalTime}) \times 100 \tag{3.2}$$

  Where $upTime$ is the time that the Web service can receive any service request, and $totalTime$ is the total measurement time, it can also be the sum of $upTime$ and $downTime$, $downTime$ is the time that the Web service is not able to receive service requests. *Availability* is measured in percentage, for example, the service *'A'* is 98% available.

- **Response Time** *Response Time* is the average time required to response a request. The metric for *Response Time* responds the question, *What is the average time on responding a request?* [45]

  The metric for *Response Time* is given by the Equation 3.3, where $T_{request}$ is the time (timestamp) when the service request is sent to the service, and $T_{response}$ is the time (timestamp) when the service response is received from the service. *Response Time* is measured in millisecond, for example, the response time for the service *'A'* is 1200 millisecond.

$$ResponseTime = T_{response} - T_{request} \tag{3.3}$$

  Response Time has become a critical quality attribute for services, because if a consumer perceives that a service takes a lot of time in response a request. It is likely that the consumer changes the service for another with less response time [43].

- **Reliability** Since there are different metrics for *Reliability*, we select the MTBF (Mean Time Between Failures), which is calculated by the Equation 3.4.

$$Reliability = \frac{\sum_1^{n-1}(FT_i - FT_{i+1})}{nFailures} \tag{3.4}$$

  Where FT_i is the Failure Time for the failure i, and $nFailures$ is the total number of failures. The unit of measure is in format time, for example, the *Reliability* (MTBF) for the service *'A'* is 17h34m10s.

- **Robustness** The metric for *Robustness* modifies the request sent to the service, in order to inject faults. Two kinds of faults were adopted for this metric:

  1. **Malformed XML**, This kind of fault injection inserts segments of malformed XML in order to produce faults in the server and exhibit relevant information about the service. For example: missing close tags, adding new elements, replacing standard tags from SOAP, adding special characters in the parameters or duplicating the request.

2. **XML Bomb Injection**, This kind of fault injection expands small messages to bigger ones, by increasing the length of the request, to produce a high load in the service and increasing the processing time, even the high memory consumption can crash the service.

The evaluation of these fault injection is made through the HTTP status code of the service response. The analysis of the status code determines what is the behavior of the service respecting to the fault. In this work, we analysis three types of status codes[1]:

1. **200 - OK**, the request was successfully responded.
2. **400 - Bad Request**, the request could not be understood by the server due to malformed syntax.
3. **500 - Internal Server**, the server encountered an unexpected condition which prevented it from fulfilling the request.

In total, *Robustness* injects six faults (five of Malformed XML and one for XML Bomb Injection). The metric is defined by the Equation 3.5, where $acceptedFaults$ are the number of fault injections not handled correctly, which are potential vulnerabilities for the service, and $nFaults$ is the total number of injected faults. The unit of measure for *Robustness* is in percentage, for example, the service *'A'* is 80% robust.

$$Robustness = (1 - \frac{acceptedFaults}{nFaults}) \times 100 \tag{3.5}$$

The Monitoring Tool

For this empirical study, the monitoring tool should be less intrusive as possible. According to Subsection 2.1.7, Dynamo, Cremona and SALMon are monitoring tools that somehow are integrated into the Web service, which can produce an increase in the processing time of a request. Additionally, they collect quality values from the service side, which means that the monitoring is from the point of view of the service. This perspective does not fit with the perspective of our experimental evaluations. On the other hand, WebInject and FlexMonitorWS are standalone applications which send requests straight to the Web service in order to collect quality values from the service responses. These monitoring tools act as a service customer, and the monitoring is done from the point of view of the customer, this perspective is the ideal for our evaluations. However, WebInject does not cover all quality attributes of our quality model, since it collect quality values only for *Response Time*.

For these reasons, we chosen FlexMonitorWS as the monitoring tool for our experimental evaluation, because it provide the necessary flexibility to create monitors with different profiles according to its feature model (Section 2.1.7).

[1]IETF internet standards: `https://www.ietf.org/`

Quality Level

In order to identify degradation in the quality level of quality attributes, it is needed to define formally what is a quality value. A quality value is a numeric value which is the result of applying a metric of a quality attribute over a Web service, in order to obtain a quantitative measurement of the quality of service respecting that quality attribute. For instance, a quality value for *Response Time* can be 25 milliseconds. Therefore, quality attributes are represented as a set $A = \{A_1, A_2, A_3, ..., A_n\}$ where $n \in \mathbb{N}^+$, and A_i is a quality attribute with $i = 1, ..., n$. Metrics are also defined for each quality attribute as $M_i = \{M_{i1}, M_{i2}, M_{i3}, ..., M_{im}\}$ where $m \in \mathbb{N}^+$, $i = 1, ..., n$, and M_{ij} is a metric for the quality attribute A_i with $j = 1, ..., m$. So a quality value is defined by the Equation 3.6.

$$Q^{value}(S, A_i, M_{ij}) = v \tag{3.6}$$

where $v \in \mathbb{R}$ is the quality value of the quality attribute A_i using the quality metric M_{ij} in the Web service S.

The quality level for a quality attribute is composed of quality values collected over a period of time. Insofar quality values present smooth variations over the time, and quality levels are represented by a range of quality values within a confidence factor. For example, the response time is greater than 15 ms and less than 45 ms for 95% of the cases. So the quality level is defined by the Equation 3.7.

$$Q^{level}(S, A_i, M_{ij}) = [v_{min}, v_{max}] \text{ for } C\% \text{ of the cases} \tag{3.7}$$

where v_{min} and v_{max} are the minimum and maximum quality values respectively, and C is the confidence factor of the quality level.

In order to identify a degradation in the quality of service, we define three types of degradation based on the classification proposed by Mairiza in Subsection 2.2.2:

- *Absolute degradation.* An absolute degradation is identified when there is not an intersection in the quality level of two different periods of time for the same quality attribute. For example, the response time for a Web service in a period *T1* is between 23.5 to 26.2 millisecond, and in a period *T2* is between 28.6 to 34.8 milliseconds. These intervals do not have an intersection, so there is an absolute degradation in the response time.

- *Relative degradation.* A relative degradation is identified when there is an intersection in the quality level of two different periods of time for the same quality attribute. For example, the reliability for a Web service in a period *T1* is between 96.4 to 98.6 percent, and in a period *T2* is between 97.5 to 99.2 percent. These intervals have an intersection, so there is a relative degradation in the reliability.

- *No degradation.* There is not a conflict when the quality level for the same quality attribute is the same in two different periods of time. For example, the availability for a Web service in a period *T1* is between 98.1 to 99.8 percent, and in a period *T2* is between 98.1 to 99.8 percent. These intervals are the same, so there is no degradation in the availability.

Once we have defined a degradation in the quality level, it is possible to define a conflicts between quality attributes. We mapped our types of degradation with the types of conflict defined in the catalogue of conflicts among NFRs (Subsection 2.2.2). *Absolute conflict*, two quality attributes are in absolute conflict when at least one of them present an absolute degradation. *Relative conflict*, two quality attributes are in relative conflict when at least one of them present a relative degradation. *Never conflict*, two quality attributes are never in conflict when they do not present any degradation.

3.3.3 Selection of Subjects

The subjects for this experimental evaluation are two Web services. The first one is a BPEL service for a Travel Agent, which invokes to three other services (FlightReservation, HotelReservation, and CarReservation services), in order to provide the information required for the users. This Web service was implemented and executed in a laboratory environment. The second one is a real-world Web service that provides country information.

3.3.4 Experiment Design

This study is composed by two experimental evaluations, more specifically an experiment and a case study, as defined in Section 3.2.

In the case of the experiment, we have implemented a BPEL composite service, as well as the three third-party services (FlightReservation, HotelReservation, and CarReservation services). We developed these services using JAX-WS library and MySQL database. After that, these services were composed in a single BPEL service named *TravelAgent* service. A complete description of the implementation is presented further in Chapter 4. On the other hand, for the case study, we have searched on the Internet for a public interface (public WSDL) of a real-world Web service. *CountryInfo* is a single public service implemented in DataFlex, which provides basic information about any country in the world. A detailed description of this service is presented further in Chapter 5.

The defined steps in this subsection are applied for both evaluations.

Step 1: Creation of Monitors

Since FlexMonitorWS tool is based on Software Product Lines and monitoring profiles, it is easy to create monitors for specific quality attributes (Subsection 2.1.7). Following the feature model of FlexMonitorWS (Figure 2.3) there are five groups of features: *Monitoring Target*, *Quality Attribute*, *Monitoring Mode*, *Monitoring Frequency* and *Notification Type*. Since, we pretend to identify conflicts between quality attributes in Web service monitoring, and the point of view is from the customer service, our monitoring target is the *Service* (see Figure 2.4) and the monitoring configuration adopted is the *Configuration 2* (See Figure 2.5). A monitor is created for each quality attribute of the Subsection 3.3.2 following the monitoring profiles in Table 3.1

Every monitoring profile has a frequency of 30 seconds and a notification by writing in a file log; results in file log would be used to analyses data in following steps.

Table 3.1: Monitoring Profiles for the Empirical Study

Monitoring Profile	Monitoring Target	Quality Attribute	Operation Mode
AccMonitor	Service	Accuracy	Invocation
AvaMonitor	Service	Availability	Invocation
ResMonitor	Service	Response Time	Invocation
RelMonitor	Service	Reliability	Invocation
RobMonitor	Service	Robustness	Invocation

Step 2: Execution

Once the monitors are created, the next step is to monitor the Web services in order to collect quality values to measure the quality level of the services. It is important to collect a representative sample of the quality values, in order to generalize the behavior of the quality level of the service. For this reason, monitoring is executed during 24 consecutive hours, since it is known that many Web systems usually perform maintenance activities in the early hours of the day. We execute the monitoring in two different scenarios, in order to evaluate degradation in the quality level of the service.

1. **Scenario 1: Monitoring in Isolation**. In this scenario, every quality attribute is monitored when no other attributes are monitored. The aim of this scenario is to build a basis state of the quality level of the Web service, and subsequently, the collected values will be compared with the values of the second scenario.

2. **Scenario 2: Monitoring in Pairs**. In this scenario, quality attributes are grouped in pairs and monitored at the same time over the same Web service. The aim of this scenario is to intend to produce a degradation in the quality level of at least one quality attribute. The following pairs of quality attributes are scheduled for monitoring:

   (a) Accuracy - Availability
   (b) Accuracy - Response Time
   (c) Accuracy - Reliability
   (d) Accuracy - Robustness
   (e) Availability - Response Time
   (f) Availability - Reliability
   (g) Availability - Robustness
   (h) Response Time - Reliability
   (i) Response Time - Robustness
   (j) Reliability - Robustness

Step 3: Data Analysis

Once we have collected a sample of the quality values in the two scenarios (monitoring in isolation and pairs), we need to summarize and generalize these values. Statistical operations are usually used to summarize samples, such as mean, median, proportion, correlation, or standard deviation. Of course, the values for a statistical operation, such as mean, will vary in different samples. In order to know the magnitude of these variations and their margin of errors in a global sense, we estimate its sampling distributions. Sampling distributions are probabilistic distributions of all possible values for a statistical operation.

Bootstrapping is a statistical method that allows estimate the sampling distribution for a sample, by making random sampling, with replacement, from the original sample [12]. The aim is to produce more samples and apply the same statistical operation to every new sample and hence draws its probabilistic distribution.

In order to summarize and generalize the collected quality values, we use the mean as statistical operation and bootstrapping to estimate the sampling distribution, respectively. Consequently, the quality level (Equation 3.7) for a quality attribute is represented by the sampling distribution using a confidence factor of 95%; it means that the minimum and maximum values for the quality level will be the values of the percentiles of 2.5 percent and 97.5 percent.

Similarly, degradation in the quality level will be determined by comparing the quality levels of the same quality attribute in the two scenarios and following the definitions in Subsection 3.3.1.

3.3.5 Validity Evaluation

Commonly, there are threats to validity in any empirical study. This section provides a brief overview of the types of threats to validity. Since we conduct two different experimental evaluations (an experiment and a case study), threats to validity will be different for each one. In this situation, they will be listed in each evaluation respectively (Chapter 4 and Chapter 5). The threats that can be found are:

- Threats to *conclusion validity* concern with the relationship of between the treatment and the outcome.
- Threats to *internal validity* concern with any confounding factor that could influence the results.
- Threats to *construct validity* concern with the relations between the theory and the observation, ensuring that the treatment reflects the cause and the outcome reflect the effect.
- Threats to *external validity* concern the possibility of generalizing our results.

3.4 Final Remarks

This chapter presented an overview of our proposed solution, which consists of two empirical evaluations, an experiment and a case study. It was also presented the definition of experimental study, as well as its main objective. A planning of the evaluations were also detailed which include our hypothesis, independent and depend ent variables, the subjects of the study, the experiment design and some threats to validity our study.

Chapter 4

Experiment: The Composite Travel Agent Service

This experiment aims to observe the changes in the quality level of a composite service when it is being monitored. Mainly, we look for degradation in the quality values for quality attributes (Subsection 3.3.2) caused by monitoring. In order to achieve this objective, we use the monitors created in Subsection 3.3.4. Following the defined steps from Subsection 3.3.4, each quality attribute is monitored in isolation to create a basis for comparison, then quality attributes are monitored in pairs, in order to identify degradation in the quality levels.

4.1 Preparation

The object of study for this experiment is a composite service (Subsection 2.1.3) named *TravelAgent* service. This service is used for searching different service during a trip, such as rent a car, book a hotel room and buy airline tickets. *TravelAgent* Service is a service orchestration composed by three third-party services: *CarReservarion* service, *FlightReservation* service and *HotelReservation* services, which are modeled using BPEL (Business Process Execution Language). Figure 4.1 shows the BPEL service model.

Once *TravelAgent* service receives a request, all parameters are validated and immediately requests for the three third-party services are created. Then the service invocations are executed sequentially. Firstly *TravelAgent* calls to *FlightReservation* service sending all parameters (`checkin_date`, `checkout_date`, `origin _country` and `destination_coun try`), *FlightReservation* service looks for all available flights from `origin_country` to `destination_country` in the `checkin_date` and the return in the `checkout_date`. Secondly, *HotelReservation* service is invoked to retrieves available rooms in hotels in the `destination_country` from `checkin_date` to `checkout_date`. Thirdly, *TravelAgent* invokes to *CarReservation* service, in order to retrieve available cars to rent in the `destination_country` from `checkin_date` to `checkout_date`. Finally, *TravelAgent* creates the final service response using the service responses from every third-party service, and then it is returned to the service customer. The information for flights, cars and hotels were randomly auto-generated in each Web service.

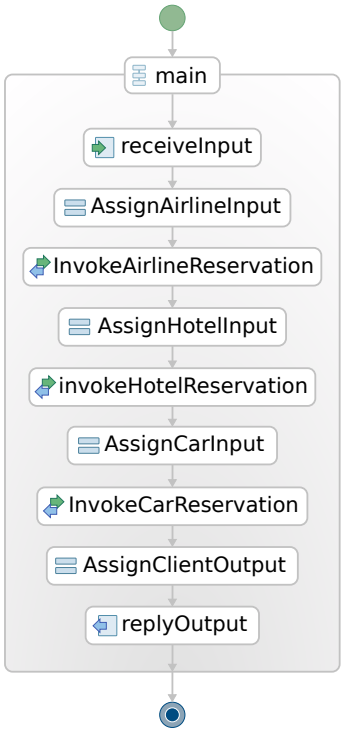


Figure 4.1: Web services BPEL model.

In order to reproduce a more real scenario, Web services were allocated in different locations and hosted in different operating systems. To achieve this configuration, we rented four virtual machines from Google Cloud Platform[1] in different locations. Figure 4.2 shows the service distribution for the composition service, and Table 4.1 describes the technical characteristics of each environment.

Figure 4.2: Web services locations.

[1] https://cloud.google.com/

Table 4.1: Technical Characteristics of Web service environments

	br-campinas-sp	**us-east1-c**
Web service	– (Monitor)	TravelAgent
Location	Campinas, SP, Brazil	South Carolina, USA
Server name	msc2015	travelagent-ws
SO	GNU/Linux	Microsoft Windows
SO Distribution	Fedora release 21	Windows Server 2012 R2 Datacenter
Number vCPU	4	1
Number Cores	2	1
Architecture	x86_64	x86_64
Processor	Intel(R) Core(TM) i3-2130 CPU @ 3.40GHz	Intel(R) Xeon(R) CPU @ 2.30GHz

	us-central1-c	**europe-west1-c**	**asia-east1-c**
Web service	FlightReservation	CarReservation	HotelReservation
Location	Iowa, USA	St. Ghislain, Belgium	Changhua County, Taiwan
Server name	flight-reservation	car-reservation	hotel-reservation
SO	GNU/Linux	GNU/Linux	GNU/Linux
SO Distribution	Ubuntu 16.04 LTS	Debian GNU/Linux 8.4 (jessie)	CentOS Linux release 7.2.1511 (Core)
Number vCPU	1	1	1
Number Cores	1	1	1
Architecture	x86_64	x86_64	x86_64
Processor	Intel(R) Xeon(R) CPU @ 2.30GHz	Intel(R) Xeon(R) CPU @ 2.50GHz	Intel(R) Xeon(R) CPU @ 2.50GHz

TravelAgent service was installed into Apache ODE [2] and deployed in Apache Tomcat[3] hosted in the server *us-east1-c*. Moreover, all third-party services were developed in JAVA using the JAX-WS library and MySQL as database, and then services were deployed in Apache Tomcat. The services were hosted in the servers `'us-central1-c'`, `'europe-west1-c'`, and `'asia-east1-c'`.

The monitors were installed in the `'br-campinas-sp'` computer (see Table 4.1) located in Sao Paulo, Brazil, and requests are sent to the *TravelAgent* service hosted in `'us-east1-c'` located in South Carolina, USA.

4.2 Execution

The experiment was executed over 24 hours for each monitoring scenario (see Subsection 3.3.4), and quality values were collected for each quality attribute (defined in Subsection 3.3.2).

Listing 4.1 shows the request message sent to the service for the monitors each 30 seconds, and Listing 4.2 shows the response message returned by the *TravelAgent* service.

Listing 4.1: TravelAgent Monitoring Request

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
   envelope/">
  <SOAP-ENV:Body>
    <reservationType xmlns="http://travelagent.soa.jael.lsd.ic.
       unicamp.br/schema/travelagent">
      <name>Jael</name>
      <origin>PER</origin>
      <destination>ARG</destination>
      <checkoutDate>2015-04-30</checkoutDate>
      <checkinDate>2015-04-27</checkinDate>
    </reservationType>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4.2: TravelAgent Monitoring Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
   envelope/">
  <soapenv:Body>
    <reservationType xmlns="http://travelagent.soa.jael.lsd.ic.
       unicamp.br/schema/travelagent"
        xmlns:tns="http://travelagent.soa.jael.lsd.ic.unicamp.br/
           schema/travelagent"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <tns:name>Jael</tns:name>
```

[2]Orchestation Director Engine which execute business process defined in WS-BPEL.

[3]A open source web server and servlet container developed by Apache Software Foundation.

```
      <tns:origin>PER</tns:origin>
      <tns:destination>ARG</tns:destination>
      <tns:departDate>2015-04-27</tns:departDate>
      <tns:arriveDate>2015-04-30</tns:arriveDate>
      <tns:roomNumber>66 - Superior Suite</tns:roomNumber>
      <tns:hotelName>InterContinental Hoteles y Resorts</
         tns:hotelName>
      <tns:airlineName>Sky Airline</tns:airlineName>
      <tns:carType>Fiat 500</tns:carType>
      <tns:flightTicketPrice>398.7</tns:flightTicketPrice>
      <tns:roomPrice>95.77</tns:roomPrice>
      <tns:carRentPrice>72.46</tns:carRentPrice>
    </reservationType>
  </soapenv:Body>
</soapenv:Envelope>
```

4.3 Data Analysis

Once we have collected a sample of the quality values for each quality attribute, data analysis is used, according to Subsection 3.3.4, to estimate the quality level of the service in each scenario.

Accuracy

Table 4.2 shows the summarization of the quality values collected by monitoring *Accuracy*. From this results, we can observe that the number of invalid responses for monitoring *Accuracy* in isolation and with *Response Time* is the same (one invalid response). In the isolated scenario, it is believed that the response was lost, because any abnormal behavior was found in log files. On the other hand, a similar case is observed for monitoring *Accuracy* with *Availability* and with *Reliability*, which present 10 and 11 invalid responses, respectively. The reason for invalid responses was due of a lack of memory. Processes in Apache ODE are loaded in-memory bare, it means only the compiled BPEL process is loaded, and the definitions are loaded in each invocation. After, they are persisted in memory for a short period of time unnecessarily. Finally, *Accuracy* with *Robustness* was the worst scenario, because it received 18 invalid responses. The cause was also for lack of memory, where fault injection (from *Robustness* monitor) has impacted on the quality of service, producing a larger number of invalid responses.

Figure 4.3 shows the sampling distribution for *Accuracy*, where the shaded area under the curve represents the 95% of confidence (from 2.5% to 97.5%). It is observed that there was not a degradation on the quality level with 95% of confidence (Subsection 3.3.1) between *Accuracy* in isolation and monitored with *Response Time*, because the quality level is the same than monitoring in isolation. However, *Accuracy* suffered an absolute degradation during monitoring in pairs with the rest of the attributes, because their quality levels do not present an intersection. The cause of degradation was for a

Table 4.2: Monitoring results for *Accuracy* in *TravelAgent* service.

	Req	VRes	IRes	Mean Quality
Isolated	1100	1099	1	99.98%
Availability	1111	1101	10	99.71%
ResponseTime	1098	1097	1	99.98%
Reliability	1102	1091	11	99.70%
Robustness	1099	1081	18	98.92%

lack of memory to process a greater amount of requests.

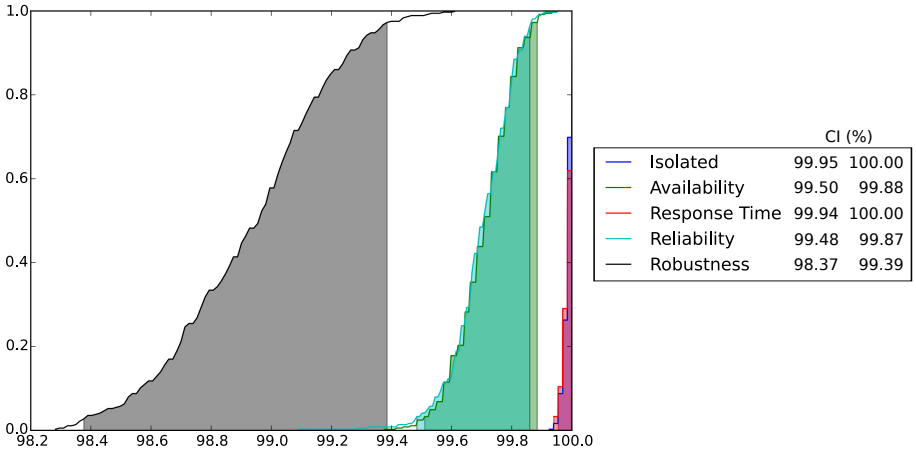


Figure 4.3: Sampling distribution for monitoring *Accuracy* in *TravelAgent* service.

Availability

Similarly to the previous quality attribute, Table 4.3 shows the summarization of the results for *Availability*. From this, we can observe that the *TravelAgent* service was 100 percent available during the isolated scenario, and in pair with *Response Time*, and *Robustness*. While monitoring *Availability* with *Accuracy* and with *Reliability*, the service did not respond to 12 requests for both scenarios. In both scenarios, the lack of memory was the reason for the unavailability of the service to process new requests.

Table 4.3: Monitoring results for *Availability* in *TravelAgent* service.

	Req	VRes	IRes	Mean Quality
Isolated	1100	1100	0	100.0%
Accuracy	1095	1083	12	99.56%
ResponseTime	1096	1096	0	100.0%
Reliability	1099	1087	12	99.30%
Robustness	1103	1103	0	100.0%

Figure 4.4 shows the sampling distributions of the quality levels for *Availability* with a 95% of confidence (shaded areas). Comparing the quality levels (according to the Subsection 3.3.1), we can observe that there was not a degradation in the *Availability* when it was monitored in isolation, with *Response Time* and with *Robustness*, because

their quality levels (confidence intervals) are identical. On the other hand, the quality level obtained for monitoring *Availability* with *Accuracy* and *Reliability*, are slightly smaller than the rest, which represents an absolute degradation of the *Availability*, because their quality levels do not present an intersection.

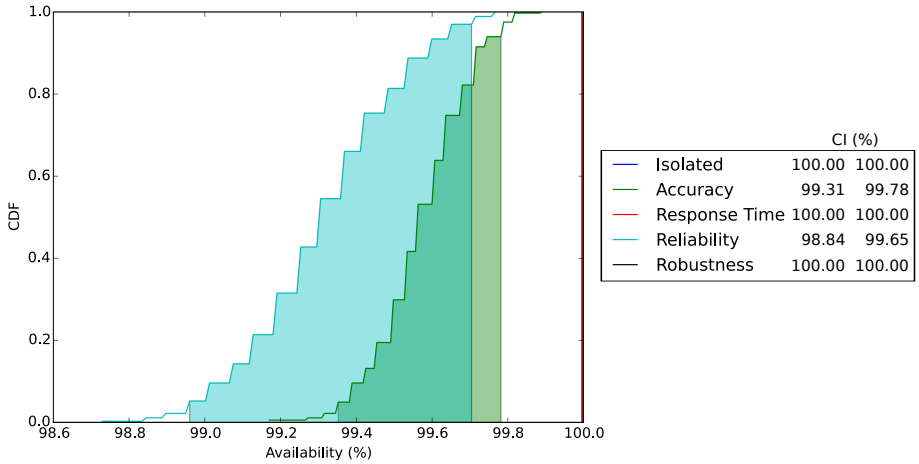


Figure 4.4: Sampling distribution for monitoring *Availability* in *TravelAgent* service.

Response Time

Since the *TravelAgent* service needs to process a greater amount of requests and invokes to the third-party services many more times, degradation in *Response Time* is more evident. Table 4.4 summarizes the result of monitor *Response Time* in every scenario. We can observe that there are no invalid responses in all scenarios, since the metric for *Response Time* does not validate the response. Additionally, we also note that the mean quality level for *Response Time* monitored in pair with *Robustness* is lower than the rest. It is believed that requests with fault injection are rejected before to be processed by the service because they are rejected by the application server. In this way, the service does not process the requests, as a consequence, requests are responded in shorter time.

Table 4.4: Monitoring results for *Response Time* in *TravelAgent* service.

	Req	**VRes**	**IRes**	**Mean Quality**
Isolated	1100	1100	0	897.64 ms
Accuracy	1097	1097	0	1224.63 ms
Availability	1095	1095	0	1162.78 ms
Reliability	1086	1086	0	1117.92 ms
Robustness	1110	1110	0	1078.32 ms

Figure 4.5 shows the sampling distribution for 365 simulations using bootstrapping over the collected quality values of monitoring with a 95% of confidence (shaded areas). There is an absolute degradation between *Response Time* monitored in isolation respect to the monitoring in pairs. On the other hand, there is a relative degradation between the results of monitoring in pair themselves, because exists an intersection between their

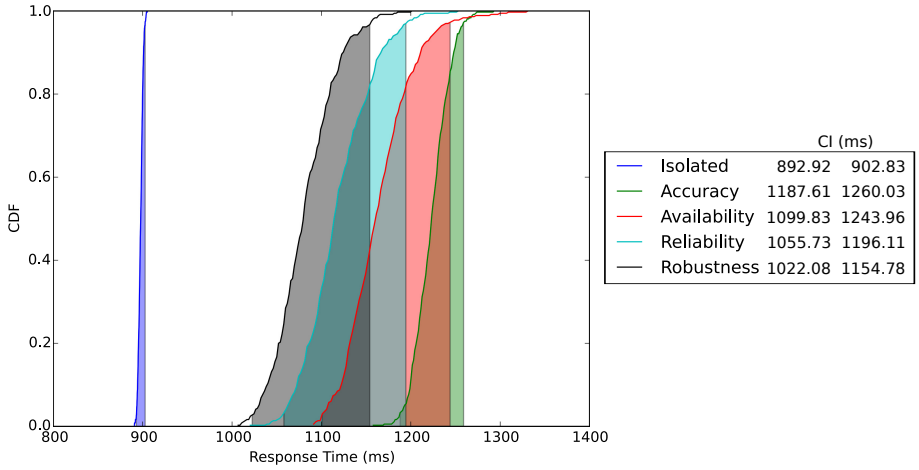


Figure 4.5: Sampling distribution for monitoring *Response Time* in *TravelAgent* service.

confidence intervals, except for monitoring *Response Time* with *Robustness* and *Accuracy*, where a quality degradation is observed.

It is possible to distinguish three scenarios which have a meaningful difference, *Response Time* monitored in isolation, monitored with *Robustness* and with *Accuracy*, where *TravelAgent* service took more time in response when it was monitored *Response Time* with *Accuracy*. This result contradicts our hypothesis about to get more delays during fault injection (*Robustness*).

Reliability

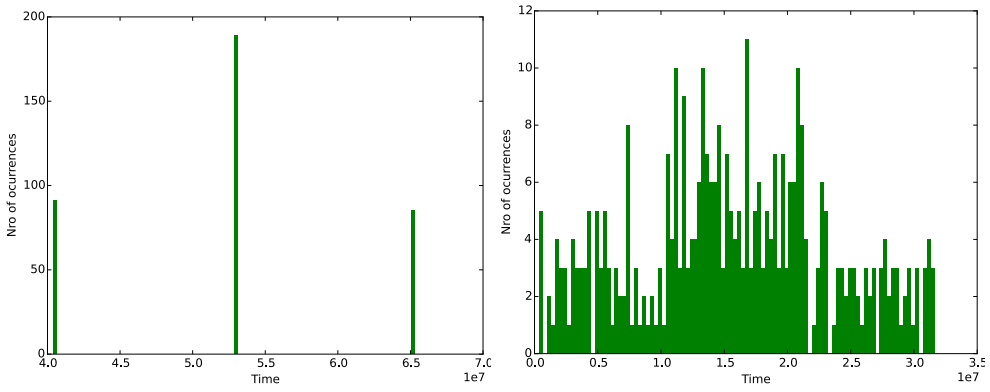
For every quality value collected by monitoring *Reliability* was computed the Mean Time Between Failures (MTBF), if more than one consecutive invalid response is detected it is considered into the same fail. Table 4.5 shows the summarization of the results for monitoring 24 hours. We can observe that the number of failures increases when *Reliability* is monitored in pairs with other attributes. It was discovered that the failures are produced for lack of memory in the server, due to the quantity of request received for the service. In contrast, the MTBF decreases which means that the time to happen a failure is increasingly smaller.

Table 4.5: Monitoring results for *Reliability* in *TravelAgent* service.

	Req	**VRes**	**IRes**	**Failures**	**Mean Quality**
Isolated	1037	1037	0	0	23h58m47s
Accuracy	1062	1056	6	3	14h40m56s
Availability	1065	1046	19	4	06h36m42s
ResponseTime	1046	1038	8	5	02h48m42s
Robustness	1050	1048	2	2	21h53m44s

In this scenario, we took a different approach to performing the sampling distribution. Smoothed bootstrap technique was applied because the number of results for the sample is too small. When bootstrap is used over a few quantity of results, the bootstrap distribution becomes too discrete because new samples are repeated many times. Smoothed

bootstrap applies a convolution method of regularization to reduce the discreteness of the first bootstrap distribution, by adding a small random noise to each bootstrap sample. Figure 4.6a shows a histogram for bootstrapping *Reliability* with 365 simulations, it presents a discrete distribution about the MTBF, which is not a realistic result. Whereas, Figure 4.6b presents a histogram for smoothed bootstrap with the same quantity of simulations. This distribution is a better representation for a realistic situation for *Reliability*.



(a) Bootstrap (b) Smoothed Bootstrap

Figure 4.6: Histogram for bootstrapping *Reliability* (MTBF) with 365 simulations.

Figure 4.7 shows the sampling distribution of *Reliability* by using 365 bootstrapping simulations with a 95% of confidence (shaded areas). A quality degradation is observed during monitoring *Reliability* with *Accuracy*, with *Availability* and with *Performance* respect to monitoring *Reliability* in isolation. Since we observe that these scenarios present an absolute degradation, because there is not intersections between their quality levels. On the other hand, a relative degradation is observed in the quality level during monitoring *Reliability* with *Robustness*, because its quality levels present an intersection.

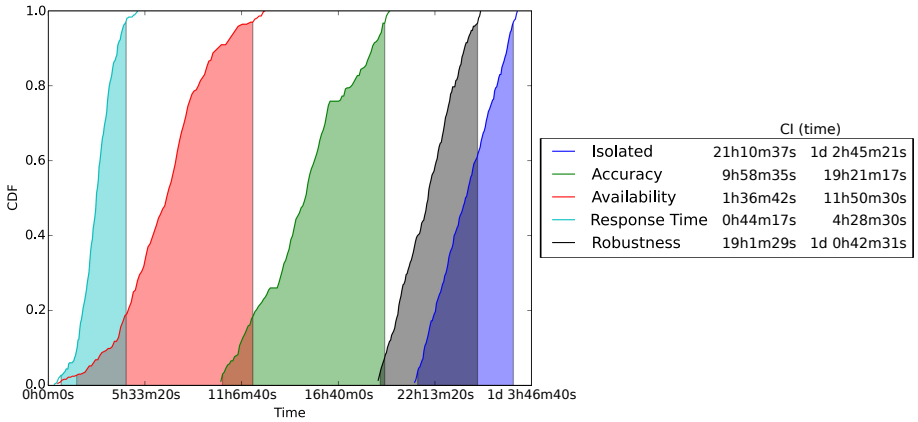


Figure 4.7: Sampling distribution for monitoring *Reliability* in *TravelAgent* service.

Robustness

For *Robustness*, variations are very few, since the service responded correctly to every request with injected faults. In the case of monitoring *Robustness* in isolation and with

Response Time, service produces wrong responses for some requests because the Web service stays inactive for a few seconds. It is believed that the service inactivity was because of the injected faults. Table 4.6 shows the results for 24 hours of monitoring. In this case, *Robustness* presents no degradations concerning monitoring in isolation or in pairs. Listing 4.3 shows the result of fault injections during monitoring, we observe that *TravelAgent* service responded appropriately for all injections.

Table 4.6: Monitoring results for *Robustness* in *TravelAgent* service.

	Req	**VRes**	**IRes**	**Mean Quality**
Isolated	1054	1053	1	99.98%
Accuracy	1063	1063	0	100.0%
Availability	1065	1065	0	100.0%
ResponseTime	1073	1064	9	99.92%
Reliability	1064	1064	0	100.0%

Listing 4.3: *Robustness* results for monitoring *TravelAgent* service

```
Robustness report - Service: TravelAgentServiceProcess
URL: http://104.196.130.132:8080/ode/processes/
    TravelAgentServiceProcess?wsdl
Script Name                    Code Expected  Code Returned   Vulnerability

Malformed XML 1 - Tags from WS    400  500        500      VNF - Passed
Malformed XML 2 - Standard tags   400  500        500      VNF - Passed
Malformed XML 3 - Special param   400  500        500      VNF - Passed
Malformed XML 4 - New parameter   400  500        500      VNF - Passed
Duplicate XML Requests            400  500        500      VNF - Passed
XML Bomb injection                400  500        500      VNF - Passed
```

Figure 4.8 shows the sampling distribution of the mean of the robustness percentage in 365 bootstrapping simulations, as well as their quality levels (confidence intervals). The difference in the quality level of *Robustness* monitoring in isolation and with *Response Time* respect to the other scenarios is very little. However, this difference represents a relative degradation of the *Robustness*.

4.4 Results

Monitoring quality attributes in pairs in *TravelAgent* service produced variations in the quality levels for each quality attribute. Consequently, we have represented conflicts between quality attributes using a two-dimensional matrix (as Table 2.3), that represents the conflicting relationship between two attributes (Subsection 3.3.1). For this, we use the conflict categorization defined in Subsection 3.3.1. *Absolute conflict* labeled by X, *Relative conflict* labeled by *, and *No conflict* labeled as 0. Table 4.7 presents the identified conflicts between quality attributes for our experiment.

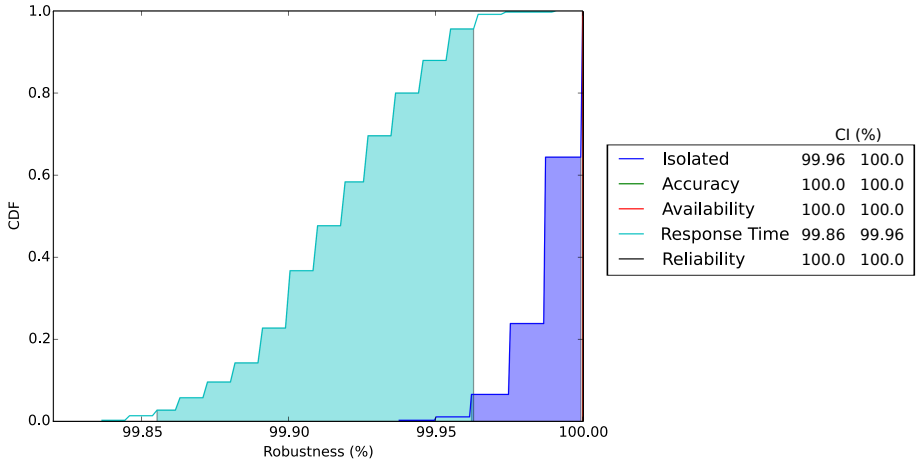


Figure 4.8: Sampling distribution for monitoring *Robustness* in *TravelAgent* service.

Table 4.7: Conflict Quality Attributes for *TravelAgent* service monitoring

	Accuracy	**Availability**	**Response Time**	**Reliability**	**Robustness**
Accuracy		x	o	x	x
Availability	x		o	x	o
Response Time	x	x		x	x
Reliability	x	x	x		*
Robustness	o	o	x	o	

4.5 Discussion of Partial Results

In this experimental evaluation, most of the cases produced an *absolute conflict* between the quality attributes during monitoring. This shows that the quality of service is not always the same, and it presents variation over the time. The produced variations in this experiment have been due to the interaction of the different methods to collect quality values from the Web service, which caused an overload in memory. The lack of memory produced the service to return invalid responses. For example, it was observed that the Web service was less accurate when it was monitored with most of the other quality attributes, getting a higher number of invalid responses. Similarly, the response time increases due to the increase in requests sent to the service, it is obvious that the service takes more time to process a higher number of requests. On the other hand, it was expected that *Robustness* produces a higher number of invalid responses, even destabilize the Web service. However, it did not produce any invalid responses, every injected fault was rejected for the application container and they were not received by the Web service. In conclusion, using different methods to collect quality values can conflict, this conflicts can compromise an overload in the resources and a degradation in the quality of the

service.

4.6 Threats to Validity

The following threats to validity were identified:

Threats to conclusion validity:

- *Bias in the results.* In order to minimize this threat, the experiment was conducted in order to be reproducible by following the guidelines proposed by Wohlin [56].

Threats to internal validity:

- *Implementation bias of the object of study.* Because *TravelAgent* service was implemented for the researcher, it is possible that the implementation can influence in the experiment. In order to minimize this threat, a second evaluation was conducted over a real-world Web service.

Threats to construct validity:

- *Representative sampling of the collected information.* In order to minimize this threat, monitoring was executed during 24 consecutive hours. In addition, bootstrapping was applied over the collected information, in order to generalize the information for a longer time of monitoring.

4.7 Final Remarks

This chapter presented the execution of an experiment according to the planning activity defined in Chapter 3. The *TravelAgent* service was implemented in a laboratory and published in the Web, in order to monitor quality levels for each quality attributes defined in Subsection 3.3.2. The aim of this experiment was to identify degradation in the quality levels of the Web service when the attributes are monitored in pairs, where most of the quality attributes presented an absolute degradation. An advantage in conduct an experiment was that we can take control over the study, and it was possible to observe the causes of the findings. For our experiment, the principal cause of degradation was the lack of memory in the server to process a greater amount of service request. In the next chapter, we will conduct a case study in order to identify degradation in quality levels for a real Web service.

Chapter 5

Case Study: Country Info Service

In this case study, the aim is to detect potential conflicts between quality attributes (Subsection 3.3.2) during its monitoring in a real Web service. We intend to identify changes in the quality level of the service, by looking for degradation on its quality values. Monitors created in Subsection 3.3.4 are used in this case study to collect quality values. Monitoring is executed in two scenarios (Subsection 3.3.4): 1) Monitoring each quality attribute in isolation and 2) Monitoring quality attributes in pairs.

5.1 Preparation

CountryInfo[1] service is a public service developed in DataFlex, a development tool for Web applications. This Web service publishes many operations to obtain information about any country in the world, such as *ListOfContinentsByName*, *CurrencyName*, *CountryName*, *FullCountryInfo*, *ListOfLanguagesByCode*, *LanguageISOCode* and so forth. Since *CountryInfo* service is a third-party service, we do not have access to the server, application server or the Web service, so *CountryInfo* service is treated as a blackbox. In contrast to *TravelAgent* service, where we had more control of the environment. Every country information is composed of an identifier code of 2 letters according to the ISO 3166-1 alpha-2, its currency, language, its capital city, continent and telephone code. The operation *FullCountryInfo* will be the monitoring target for this case study, because it returns a complete information for a country.

5.2 Execution

The case study was conducted by monitoring *CountryInfo* service over 24 hours. Actually, quality attributes defined in Subsection 3.3.2 are monitored in two scenarios (Subsection 3.3.4): monitoring quality attributes in isolation, and monitoring quality attributes in pairs.

The Listing 5.1 shows the request sent to the service, and Listing 5.2 is the returned response.

[1] `http://www.oorsprong.org/websamples.countryinfo/CountryInfoService.wso`

Listing 5.1: CountryInfo Monitoring Request

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
   envelope/">
  <SOAP-ENV:Body>
    <FullCountryInfo xmlns="http://www.oorsprong.org/websamples.
       countryinfo">
      <sCountryISOCode>BR</sCountryISOCode>
    </FullCountryInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 5.2: CountryInfo Monitoring Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope
   /">
  <soap:Body>
    <m:FullCountryInfoResponse xmlns:m="http://www.oorsprong.org/
       websamples.countryinfo">
          <m:FullCountryInfoResult>
            <m:sISOCode>BR</m:sISOCode>
                <m:sName>Brazil</m:sName>
                <m:sCapitalCity>Brasilia</m:sCapitalCity>
                <m:sPhoneCode>55</m:sPhoneCode>
                <m:sContinentCode>AM</m:sContinentCode>
                <m:sCurrencyISOCode>BRL</m:sCurrencyISOCode>
                <m:sCountryFlag>http://www.oorsprong.org/WebSamples
                   .CountryInfo/Images/Brazil.jpg</m:sCountryFlag>
                <m:Languages>
                  <m:tLanguage>
                    <m:sISOCode>por</m:sISOCode>
                    <m:sName>Portuguese</m:sName>
                  </m:tLanguage>
                </m:Languages>
          </m:FullCountryInfoResult>
        </m:FullCountryInfoResponse>
  </soap:Body>
</soap:Envelope>
```

5.3 Data Analysis

In order to discover potential conflicts between quality attributes during monitoring, we estimate a sampling distribution for quality attributes in each scenario, as defined in Subsection 3.3.4.

Accuracy

In this case, we obtained a uniform quality level of *Accuracy* during monitoring in pairs, except for an invalid response in the isolation scenario. Table 5.1 summarize the results for each scenario, where a significant degradation in the quality level of *Accuracy* is not observed because every scenario is qualified as 100% accurate.

Table 5.1: Monitoring results for *Accuracy* in *CountryInfo* service.

	Req	VRes	IRes	Mean Quality
Isolated	1139	1138	1	99.98%
Availability	1139	1139	0	100.0%
Response Time	1140	1140	0	100.0%
Reliability	1139	1039	0	100.0%
Robustness	1137	1137	0	100.0%

In order to generalize the quality level for *CountryInfo* service. Bootstrapping is used as defined in Subsection 3.3.4. Figure 5.1 shows the sampling distribution for monitoring *Accuracy*, and we also calculate the quality level (confidence interval) for each scenario. We observe a relative degradation in all scenarios respecting the isolated scenario, since the quality level for the isolated scenario intersects with the other scenarios.

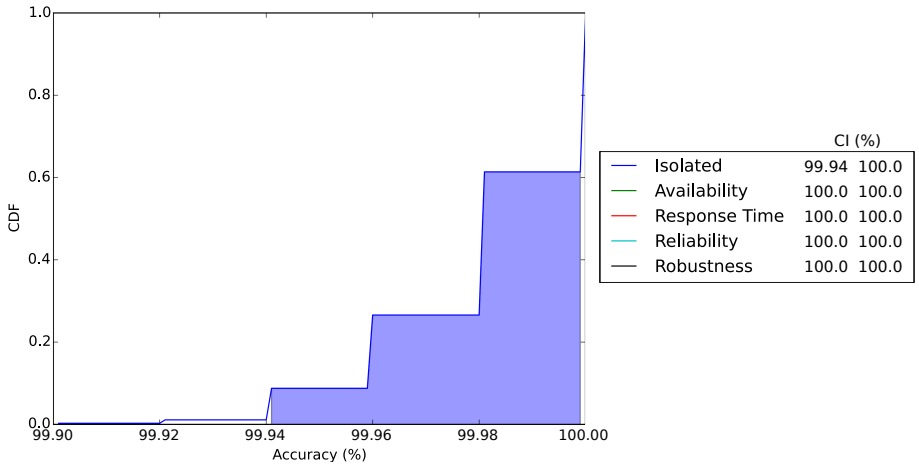


Figure 5.1: Sampling distribution for monitoring *Accuracy* in *CountryInfo* service.

Availability

Table 5.2 shows the summarization of the results for each monitoring scenario. From this, we can observe a degradation in the *Availability* when it was monitored with *Reliability*. On the other hand, we also observe that monitoring in pairs with *Accuracy* and with *Robustness* had better results than isolation scenario. Analyzing invalid responses, we detected that no response was received from *CountryInfo* service in the other scenarios.

Figure 5.2 shows the sampling distribution for *Availability* by using bootstrapping in the collected quality values. In the results, we observe that there are relative degradation in all quality attributes, since there are intersections between their quality levels (confidence intervals) in all scenarios.

Table 5.2: Monitoring results for *Availability* in *CountryInfo* service.

	Req	VRes	IRes	Mean Quality
Isolated	1140	1139	1	99.91%
Accuracy	1139	1139	0	100.0%
Response Time	1137	1136	1	99.93%
Reliability	1127	1124	3	99.82%
Robustness	1137	1137	0	100.0%

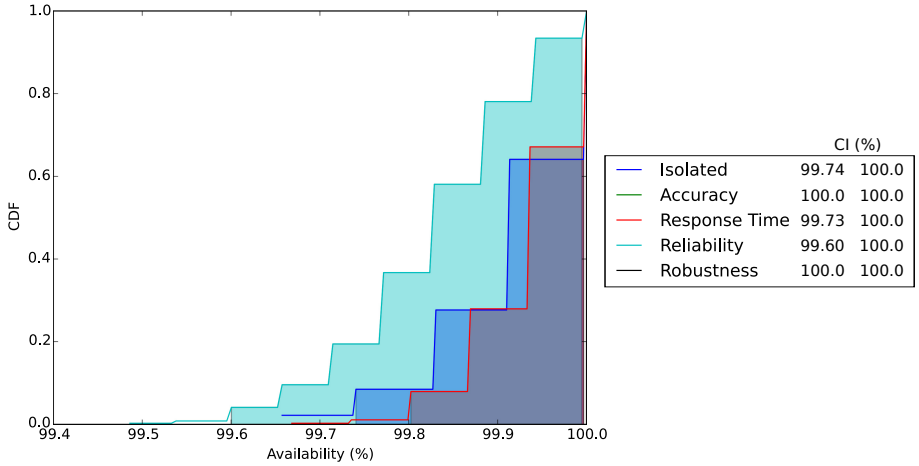


Figure 5.2: Sampling distribution for monitoring *Availability* in *CountryInfo* service.

Response Time

Table 5.3 summarize the results for 24 hours of monitoring *Response Time* in each scenario. From the results, we observe that there is a small degradation in the quality level between the scenarios, where *CountryInfo* service takes more time in response a request for monitoring in pairs. However, we also observe that it takes a shorter time during monitoring *Response Time* with *Accuracy* than the isolation scenario. Since we do not have any control over *CountryInfo* service, it was impossible for us to determine the reasons about this scenario.

Table 5.3: Monitoring results for *Response Time* in *CountryInfo* service.

	Req	VRes	IRes	Mean Quality
Isolated	1140	1140	0	317.32 ms
Accuracy	1140	1140	0	314.54 ms
Availability	1137	1136	1	325.15 ms
Reliability	1140	1140	0	318.67 ms
Robustness	1140	1139	1	320.78 ms

In Figure 5.3 is observed the sampling distribution for *Response Time*. We can observe intersections between the quality levels (confidence intervals) for monitoring *Response Time* with every other quality attribute, except *Availability*. Therefore, there is a relative degradation in these cases. On the other hand, we observe an absolute degradation in the quality level for *Response Time* monitored with *Availability*. It is suspected that since *Availability* monitor only checks if the service is available, a greater amount of requests

are sent to the service, requiring a longer time for processing. But it is refused since the number of request for *Availability* is lower than the isolated scenario (Table 5.2).

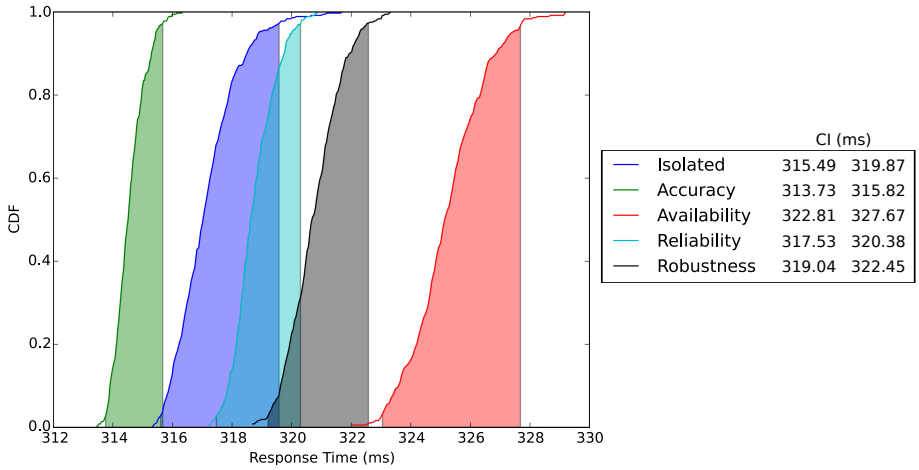


Figure 5.3: Sampling distribution for monitoring *Response Time* in *CountryInfo* service.

Reliability

For *Reliability*, the Mean Time Between Failures is computed by counting invalid responses, if one or more invalid responses are consecutive, they are considered as the same failure. Table 5.4 shows the results for 24 hours monitoring *Reliability*, and we can observe that no invalid responses were received in all scenarios, which means there is no degradation in the quality level for *Reliability*.

Table 5.4: Monitoring results for *Reliability* in *CountryInfo* service.

	Req	VRes	IRes	Failures	Mean Quality
Isolated	1099	1092	0	0	23h59m56s
Accuracy	1094	1094	0	0	23h58m11s
Availability	1095	1046	0	0	23h58m35s
Response Time	1096	1096	0	0	23h58m47s
Robustness	1095	1095	0	0	23h59m21s

Similar to Subsection 4.3, we perform a different bootstrapping technique for *Reliability*. Since the collected information is too small, smoothed bootstrapping is used to re-sampling the MTBF for *CountryInfo* service. Figure 5.4 shows the sampling distribution for *Reliability* by using smoothed bootstrapping, where we performed 365 simulations (samples for one year). The sampling distribution of the quality level shows no degradation in the all monitoring scenarios. It means that there are no conflicts between *Reliability* with the other attributes.

Robustness

Since *Robustness* measures the ability of the Web service to faces errors at runtime, the monitor for *Robustness* injects six kinds of faults in the request message (defined in Subsection 3.3.2). Listing 5.3 shows the result of fault injections during monitoring for a request

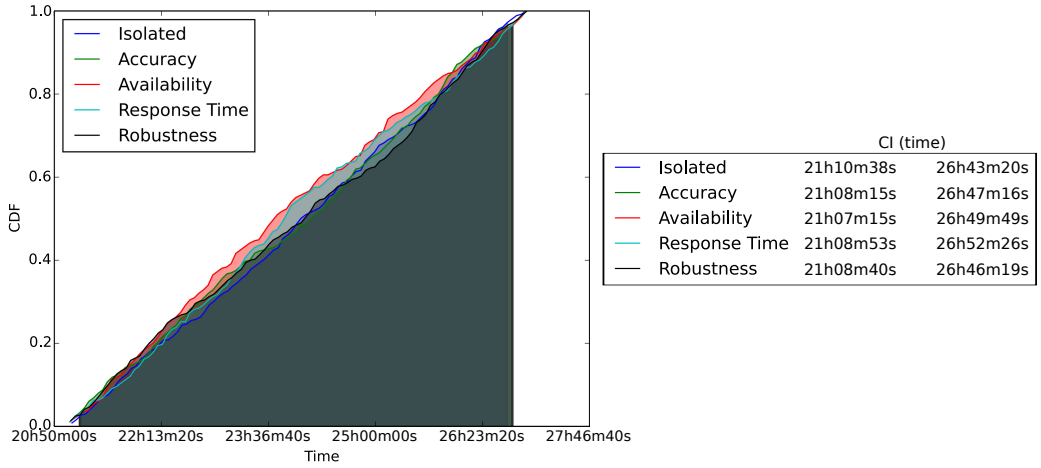


Figure 5.4: Sampling distribution for monitoring *Reliability* in *CountryInfo* service.

in *CountryInfo* service. We can observe that *CountryInfo* service responded appropriately for all the injections, except for `Duplicate XML requests`, where a duplicated request was consider as a valid request for *CountryInfo* service. Table 5.5 shows the results for 24 hours of monitoring *Robustness* in all the scenarios. The incorrect treatment for duplicate requests injection is present in all the collected information, which resulted in an 83.33% of *Robustness*. In the case of monitoring *Robustness* with *Accuracy*, the service did not respond to one request (because it was unavailable) which produce that the measurement for that scenario is lower than the others.

Listing 5.3: *Robustness* results for monitoring *CountryInfo* service

```
Robustness report - Service: CountryInfoService
URL: http://webservices.oorsprong.org/websamples.countryinfo/
    CountryInfoService.wso?wsdl
Script Name                    Code Expected  Code Returned   Vulnerability

Malformed XML 1 - Tags from WS    400  500        500       VNF - Passed
Malformed XML 2 - Standard tags   400  500        500       VNF - Passed
Malformed XML 3 - Special param   400  500        500       VNF - Passed
Malformed XML 4 - New parameter   400  500        500       VNF - Passed
Duplicate XML Requests            400  500        200       VF - FAILED
XML Bomb injection                400  500        500       VNF - Passed
```

Table 5.5: Monitoring results for *Robustness* in *CountryInfo* service.

	Req	VRes	IRes	Mean Quality
Isolated	941	0	941	83.33%
Accuracy	1090	0	1090	83.29%
Availability	1088	0	1088	83.33%
Response Time	1093	0	1093	83.33%
Reliability	1078	0	1078	83.33%

To generalize the results, we performed 365 bootstrapping executions, computing the mean for each execution. Figure 5.5 shows the sampling distribution for *Robustness*

in every scenario of monitoring. We observe that there are not degradation in all the scenarios, since there are no difference between their quality levels. We can conclude that *Robustness* does not have conflict with any other quality attribute during monitoring.

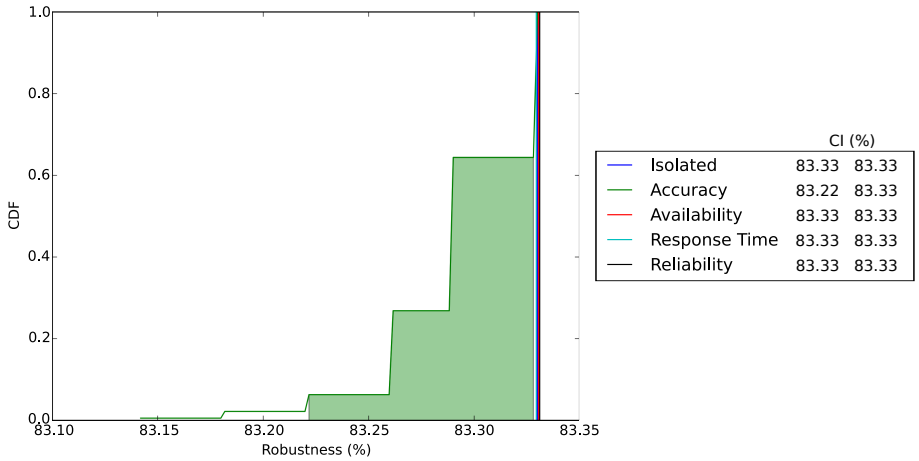


Figure 5.5: Sampling distribution for monitoring *Robustness* in *CountryInfo* service.

5.4 Results

In this case study, despite *CountryInfo* service is a simpler service than the *TravelAgent* service, we have observed degradation in the quality level. As a result, we have represented the conflict mapping for this case study using a two-dimensional matrix (as Table 2.3). Table 5.6 shows the conflict relationship between each quality attribute during monitoring of the *CountryInfo* service. For this, we use the conflict categorization defined in Subsection 3.3.1. *Absolute conflict* labeled by X, *Relative conflict* labeled by *, and *No conflict* labeled as 0.

Table 5.6: Conflict Quality Attributes for *CountryInfo* service monitoring

	Accuracy	**Availability**	**Response Time**	**Reliability**	**Robustness**
Accuracy		o	o	o	o
Availability	o		*	*	o
Response Time	*	x		*	*
Reliability	o	o	o		o
Robustness	*	o	o	o	

5.5 Discussion of Partial Results

In contrast with the results of our previous experimental evaluation, there are less conflicts between attributes in a real-world Web service. While an absolute conflict was observed between the *Accuracy* with each other attribute, in this case study, no conflicts were detected. Additional, *Response Time* presented absolute conflicts with the other attributes in the previous case. In this case, the conflict is relative which means that it is possible to no have conflicts. On the other hand, *Robustness* showed a similar behaviour to our experiment (Chapter 4), with no conflicts. We believe that Country Info service is prepared to handle the conflicts between the quality attributes, since it has many user and it attempts a huge number of request every day. However, the case study presented some relative conflicts which shows that conflict between quality attribute exists, and the quality of service can suffer negative variations.

5.6 Threats to Validity

The following threats to validity were identified:

Threats to conclusion validity:

- *Bias in the results.* In order to minimize this threat, the experiment was conducted in order to be reproducible by following the guidelines purposed by Wohlin [56].
- *Influence of external factors.* Because the case study was conducted over a public Web service, it is possible that external factor can influence in the quality level of the service. In order to minimize this threat, the Web service was re-monitored in any abnormal behavior.

Threats to construct validity:

- *Representative sampling of the collected information.* In order to minimize this threat, monitoring was executed during 24 consecutive hours, in addition, bootstrapping was applied over the collected information, in order to generalize the information for a longer time of monitoring.

5.7 Final Remarks

The chapter presented the execution of a case study according to the planning activity defined in Chapter 3. The *CountryInfo* service was chosen for this study, which we could not have any access or control. The aim of this case study was to identify degradation in the quality levels for a public Web service when the attributes are monitored in pairs. Results presented that only one quality attribute presented an absolute degradation and the others present a relative conflict or no conflict.

Chapter 6

Analysis and Discussion

The discussion of this study is addressed in the conflicts between quality attributes during monitoring Web services. For this aim, we have conducted two evaluations, an experiment and a case study, in order to construct a mapping of the identified conflicts.

6.1 Overview

Results from the experiment case (Table 4.7) have showed that monitoring quality attributes in pairs, produce quality degradation in the Web services. In this case, the cause of the quality degradation is due to the number of invalid responses, more specifically, due to the lack of memory on the server side to respond a large number of request. So it is recommended to concern about the allocation of resources during the development of Web services.

Results of the case study (Table 5.6), in contrast to Table 4.7, have showed a smaller number of conflicts between quality attributes. We believe that real-world Web services present more resources to deal with a large number of request. Nevertheless, the difference between the conflicts identified in Web services monitoring depends on various factors: 1) the implementation of the service, 2) resources allocation, 3) quality metrics definition, and 4) monitoring frequency.

6.2 Conflict Mapping

A consolidated conflict mapping between quality attributes is shown in Table 6.1, where an *absolute conflict* is declared only if both mappings identified an absolute conflict, a *relative conflict* is defined when at least one evaluation declared a relative conflict, and *no conflict* is stated when both evaluations no reported conflict.

From the consolidated conflict mapping, we can adjudge that, *Response Time* and *Reliability* are the most affected attributes because they have their quality level degraded during monitoring time with every other attribute. On the other hand, *Accuracy* is the most influential attribute, since every other attribute monitored in pairs with *Accuracy*, has its quality level degraded.

Table 6.1: Conflict Mapping between Quality Attributes during Web service monitoring

	Accuracy	**Availability**	**Response Time**	**Reliability**	**Robustness**
Accuracy		*	o	*	*
Availability	*		*	*	o
Response Time	*	x		*	*
Reliability	*	*	*		*
Robustness	*	o	*	o	

6.3 Discussion

As we have discussed in Section 2.2, conflicts between quality attributes are studied during the requirements specification stage of the software development lifecycle. Subsection 2.2.2 presented a catalogue of conflicts between Non-Functional Requirements, purposed by Mairiza and Zowghi. Table 6.2 shows a fragment of the catalogue with the same quality attributes of our study.

Table 6.2: Catalogue of Conflicts Among NFRs [33]

	Accuracy	**Availability**	**Response Time**	**Reliability**	**Robustness**
Accuracy	o		x	o	
Availability			x		
Response Time	x	x	*	*	
Reliability	o		*	o	
Robustness					

By comparing Tables 6.1 and 6.2, it is observed that *Response Time* is the most conflicting quality attribute. On the other hand, an *absolute conflict* between *Accuracy* and *Response Time* was declared in the catalogue, while our study has not reported any conflict between them. The same case for the attributes *Accuracy* and *Reliability*, any conflict was stated in the catalogue; nonetheless, a *relative conflict* was identified in our study. Additionally, the catalogue does not present information about the conflict in *Robustness*, but our study shows *relative conflicts* with *Accuracy* and *Response Time*.

The results from our experimental evaluation could be incorporated in the catalogue, but we must consider that the catalogue was built for Non-Functional Requirements and not for monitoring systems at runtime. However, it is clear that more experimental

evaluations need to be done with different quality attributes.

Chapter 7

Conclusions and Future Work

This chapter shows the conclusions of this dissertation, the main contributions and future work.

7.1 Conclusions

This dissertation has presented two experimental evaluations, an experiment and a case study, for Web services monitoring, with the aim to identify potential conflicts between quality attributes. Firstly, we selected a set of quality attributes which are observable and measurable in Web service at runtime (*Accuracy*, *Availability*, *Response Time*, *Reliability*, and *Robustness*). After that, we defined metrics for each quality attributes based on proposed metrics in the literature. Then, we looked for a non-intrusive monitoring tool which implements the defined metrics. FlexMonitorWS was selected for our study because it provides the flexibility to add new features (e.g. metrics).

For the experiment case, we implemented a BPEL composite service for a Travel Agent, and a public real Web service was used for the case study. In both evaluations, the Web service monitoring was run during 24 hours in two scenarios: 1) monitoring each quality attribute in isolation, and 2) monitoring attributes in pairs. The results showed that quality degradation is presented during monitoring quality attributes in pairs. The principal findings of this study are:

- *Response Time* is the most conflicting attribute since it presented quality degradation during monitoring with all the other attributes.
- Fault injection is the most intrusive technique, subjecting the service under stress condition or disabling it. Today real Web services are robust because they implement techniques to deal with fault.
- Strategies to collect quality values can become intrusive in the Web service generating degradation in the quality of service.
- The frequency of monitoring is another factor which generates conflicts between quality attributes, If sending requests to the Web service is more frequent, the service will stay overloaded.

- The Web service implementation can be a factor to prevent and produce quality degradation if a service is implemented without considering Non-Functional Requirements; quality attributes prone to conflict and present quality degradation.

7.2 Publications

An article has been published describing the evaluation of a case study for conflicts between the quality attributes: *Response Time* and *Accuracy*. The article was published in the journal *Electronic Notes in Theoretical Computer Science*.

- Jael Zela, and Cecília M.F. Rubira. "Quality of Service Conflict During Web Service Monitoring: A Case Study". In Electronic Notes in Theoretical Computer Science, Volume 321, Pages 113-127, 2016.

7.3 Future Work

In the study of these two experimental evaluations, we have observed some opportunities to improve Web service monitoring. We define them as future work:

- Discover conflicts in more overloaded scenarios, monitoring more than two quality attributes and using different monitoring modes, different monitoring targets and different frequencies.

- Discover conflicts in RESTful Web services and compares the results with the results of our study

- Evaluate conflicts in dynamic Web services, since our case studies were over static Web service, services are not connected and disconnected at runtime. This kind of scenario can present higher variations in the quality of the service and be prone to more conflicts between services.

- Use the conflict mapping between quality attributes in the implementation of an adaptive monitoring system. Since we can detect potential conflicts in the quality of Web services, an adaptive monitoring system can take advantage of this information in order to change its configuration at runtime. If a quality degradation is detected, the adaptive system should change the frequency, the monitoring mode or stop the monitoring.

Bibliography

[1] David Ameller and Xavier Franch. Service level agreement monitor (salmon). In *International Conference on Composition-Based Software Systems*, ICCBSS 2008, pages 224–227, Madrid, Spain, February 2008. IEEE.

[2] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg. Web services architecture requirements. Available in: `https://www.w3.org/TR/wsa-reqs/`, February 2004. [Accessed on 27/07/2016].

[3] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, October 2004.

[4] Zain Balfagih and Mohd Fadzil Hassan. Quality model for web services from multi-stakeholders' perspective. In *Proceedings of the 2009 International Conference on Information Management and Engineering*, ICIME 2009, pages 287–291, Washington, DC, USA, June 2009. IEEE Computer Society.

[5] Mario Barbacci, Mark H. Klein, Tomas A. Longstaff, and Charles B. Weinstock. Quality Attributes. Technical report, Carnegie Mellon University, December 1995.

[6] Luciano Baresi and Sam Guinea. Towards dynamic monitoring of ws-bpel processes. In *Proceedings of the 3rd International Conference on Service-Oriented Computing*, ICSOC 2005, pages 269–282, Berlin, Heidelberg, December 2005. Springer-Verlag.

[7] Barry Boehm and Hoh In. Identifying quality-requirement conflicts. In *Proceedings of the Second International Conference on Requirements Engineering*, ICRE 1996, page 218, Colorado Springs, CO, USA, April 1996. IEEE.

[8] Peter Brittenham. Understanding the WS-I Test Tools. Technical report, Web Service Interoperability Organization, 2003.

[9] Oscar Cabrera and Xavier Franch. A quality model for analysing web service monitoring tools. In *The Sixth International Conference on Research Challenges in Information Science*, RCIS 2012, pages 1–12, Valencia, Spain, May 2012.

[10] Cheol Rim Choi and Hwa Young Jeong. A broker-based quality evaluation system for service selection according to the qos preferences of users. *Information Sciences*, 277(0):553 – 566, September 2014.

[11] Nelly Delgado, Ann Quiroz Gates, and Steve Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on Software Engineering*, 30(12):859–872, December 2004.

[12] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, 1994.

[13] Alexander Egyed and Paul Grunbacher. Identifying requirements conflicts and cooperation: How quality attributes and automated traceability can help. *Software, IEEE*, 21(6):50–58, November 2004.

[14] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000.

[15] Rômulo Franco. FlexMonitorWS: Uma Solução para Monitoração de Serviços Web com foco em Atributos de QoS. Master's thesis, Institute of Computing, University of Campinas, Campinas, Sao Paulo, Brazil, 2014.

[16] Rômulo Franco, Cecilia Rubira, and Amanda Nascimento. FlexMonitorWS: Uma Solução para Monitoração de Serviços Web com foco em Atributos de QoS. In *Congresso Brasileiro de Software: Teoria e Prática, 21th Sessão de Ferramentas*, volume 2, pages 101–108, September 2014.

[17] Apache Software Fundation. Web/http test & monitoring tool. Available in: `http://axis.apache.org`, November 2011. [Accessed on 11/11/2016].

[18] Corey Goldberg. Web/http test & monitoring tool. Available in: `http://www.webinject.org`, October 2011. [Accessed on 11/11/2016].

[19] Kennet Henningsson. Trade-offs and Conflicts between Quality Attributes: A Study of Academic and Industry. Master's thesis, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Karlskrona, Sweden, 2001.

[20] Hoh In, Barry Boehm, Thomas Rodgers, and Michael Deutsch. Applying winwin to quality requirements: A case study. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE 2001, pages 555–564, Toronto, ON, Canada, May 2001.

[21] Fuyuki Ishikawa. Qos-based service selection. In Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, editors, *Web Services Foundations*, pages 375–397. Springer, September 2014.

[22] ISO/IEC. ISO/IEC 25010 - Systems and Software Engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, 2010.

[23] ISO/IEC/IEEE. Systems and software engineering – vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418, December 2010.

[24] Joao Cadamuro Junior and Douglas Renaux. Efficient monitoring of embedded real-time systems. In *Fifth International Conference on Information Technology: New Generations*, ITNG 2008, pages 651–656, April 2008.

[25] Fenil A. Khatiwala and Chirag S. Thaker. A systematic qos appraisal in soa driven systems. In *International Conference on Intelligent Systems and Data Processing*, ICISD 2011, pages 145–148, January 2011.

[26] Smita Kumari and Santanu Kumar Rath. Performance comparison of soap and rest based web services for enterprise application integration. In *International Conference on Advances in Computing, Communications and Informatics*, ICACCI 2015, pages 1656–1660, August 2015.

[27] E. M. Kyle-Bowlsbey and D. R. Zaret. Quantifying trust: Data integrity metrics. In *IEEE Military Communications Conference*, MILCOM 2005, pages 3142–3147 Vol. 5, October 2005.

[28] Mohamad Ibrahim Ladan. Web services metrics: A survey and a classification. In *International Conference on Network and Electronics Engineering*, ICNEE 2011, pages 93–98, Singapore, September 2011. IACSIT Press.

[29] KangChan Lee, JongHong Jeon, WonSeok Lee, Seong-Ho Jeong, and Sang-Won Park. QoS for Web Services: Requirements and Possible Approaches. Available in: `http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/`, November 2003. [Accessed on 12/05/2016].

[30] Heiko Ludwig, Asit Dan, and Robert Kearney. Cremona: An architecture and library for creation and monitoring of ws-agreements. In *Proceedings of the 2nd International Conference on Service Oriented Computing*, ICSOC 2004, pages 65–74, New York, NY, USA, November 2004. ACM.

[31] Lars Lundberg, Daniel Häggander, and Wolfgang Diestelkamp. Conflicts and trade-offs between software performance and maintainability. In *Performance Engineering, State of the Art and Current Trends*, pages 56–67, London, UK, UK, May 2001. Springer-Verlag.

[32] Dewi Mairiza and Didar Zowghi. An ontological framework to manage the relative conflicts between security and usability requirements. In *Third International Workshop on Managing Requirements Knowledge*, MARK 2010, pages 1–6, Sidney, Australia, September 2010.

[33] Dewi Mairiza and Didar Zowghi. Constructing a catalogue of conflicts among non-functional requirements. In *Evaluation of Novel Approaches to Software Engineering*, volume 230 of *ENASE 2010*, pages 31–44. Springer Berlin Heidelberg, 2011.

[34] Dewi Mairiza, Didar Zowghi, and Vincenzo Gervasi. Conflict characterization and analysis of non functional requirements: An experimental approach. In *IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques*, SoMeT 2013, pages 83–91, September 2013.

[35] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. Managing conflicts among non-functional requirements. In *Proceeding of the 12th Australian Workshop on Requirements Engineering*, AWRE 2009, pages 11–19, Sydney, Australia, 2009.

[36] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC 2010, pages 311–317, Sierre, Switzerland, March 2010. ACM.

[37] Anbazhagan Mani and Arun Nagarajan. Understanding Quality of Service for Web Services. Technical report, IBM, 01 2002.

[38] Andreas Metzger, Salima Benbernou, Manuel Carro, Maha Driss, Gabor Kecskemeti, Raman Kazhamiakin, Kyriakos Krytikos, Andrea Mocci, Elisabetta Di Nitto, Branimir Wetzstein, and Fabrizio Silvestri. Analytical quality assurance. In MikeP. Papazoglou, Klaus Pohl, Michael Parkin, and Andreas Metzger, editors, *Service Research Challenges and Solutions for the Future Internet*, volume 6500 of *Lecture Notes in Computer Science*, pages 209–270. Springer Berlin Heidelberg, 2010.

[39] Marc Oriol, Jordi Marco, and Xavier Franch. Quality models for web services: A systematic mapping. *Information and Software Technology*, 56(10):1167–1182, October 2014.

[40] Meysam Ahmadi Oskooei, Salwani Binti Mohd Daud, and Fang Fang Chua. Modeling quality attributes and metrics for web service selection. *Proceedings of the 3rd International Conference on Mathematical Sciences*, 1602:945–952, December 2013.

[41] Meysam Ahmadi Oskooei and Salwani Mohd Daud. Quality of service (qos) model for web service selection. In *International Conference on Computer, Communications, and Control Technology*, I4CT 2014, pages 266–270, Langkawi, Malaysia, September 2014.

[42] Mike Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: Approaches, technologies and research issues. *The International Journal on Very Large Data Bases*, 16(3):389–415, July 2007.

[43] Ramakrishnan Rajamony and Mootaz Elnozahy. Measuring client-perceived response times on the www. In *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems*, volume 3 of *USITS 2001*, pages 16–16, Berkeley, CA, USA, 2001. USENIX Association.

[44] T. Rajendran and P. Balasubramanie. Analysis on the study of qos-aware web services discovery. *Journal of Computing*, 1:119–130, December 2009.

[45] Shuping Ran. A model for web services discovery with qos. *ACM Special Imterest Group on E-Commerce Exchanges*, 4(1):1–10, March 2003.

[46] Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic qos and soft contracts for transaction-based web services orchestrations. *IEEE Transactions on Services Computing*, 1(4):187–200, October 2008.

[47] Ulrich Schmid. Monitoring distributed real-time systems. *Real-Time Systems*, 7(1):33–56, 1994.

[48] Quan Z. Sheng, Xiaoqiang Qiao, Athanasios V. Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218 – 238, 2014.

[49] Alberto Sillitti, Angelo Gaeta, Antonio De Nigro, Debora Desideri, Francisco Garijo, Franciso Perez Sorrosal, Jose M. Cantera, Katharina Mehner, Marcos Reyes, Mike Fisher, Nikolaos Tsouroulas, Pascal Bisson, Piero Corte, Raffaele Piccolo, Ricardo Jimenez, Sven Abels, and Yosu Gorronogoitia. NEXOF-RA: NESSI Open Framework – Reference Architecture. Technical report, Networked European Software and Services Initiative, December 2009.

[50] Microsoft Patterns & Practices Team. *Microsoft Application Architecture Guide, 2nd Edition*. Microsoft Corporation, 2009.

[51] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79:70–85, January 2014.

[52] Qianxiang Wang, Yonggang Liu, Min Li, and Hong Mei. An online monitoring approach for web services. In *The 31th IEEE International Conference on Computers, Software and Applications*, volume 1 of *COMPSAC 2007*, pages 335–342, Beijing, China, July 2007.

[53] Qianxiang Wang, Jin Shao, Fang Deng, Yonggang Liu, Min Li, Jun Han, and Hong Mei. An online monitoring approach for web service requirements. *IEEE Transactions on Services Computing*, 2(4):338–351, October 2009.

[54] Qianxiang Wang, Jin Shao, Fang Deng, Yonggang Liu, Min Li, Jun Han, and Hong Mei. An online monitoring approach for web service requirements. *IEEE Transactions on Services Computing*, 2(4):338–351, October 2009.

[55] Neustar Webmetrics. Web/http test & monitoring tool. Available in: `http://www.webmetrics.com`, November 2011. [Accessed on 11/11/2016].

[56] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[57] Norazlin Yusop, Didar Zowghi, and David Lowe. The impacts of non-functional requirements in web system projects. *International Journal of Value Chain Management*, 2(1):18–32, 2008.

[58] Zibin Zheng and Michael Lyu. Qos evaluation of web services. In *QoS Management of Web Services*, pages 19–39, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[59] Zibin Zheng, Yilei Zhang, and Michael Lyu. Distributed qos evaluation for real-world web services. In *IEEE International Conference on Web Services*, ICWS 2010, pages 83–90, July 2010.

[60] Zibin Zheng, Yilei Zhang, and Michael Lyu. Investigating qos of real-world web services. *IEEE Transactions on Services Computing*, 7(1):32–39, January 2014.